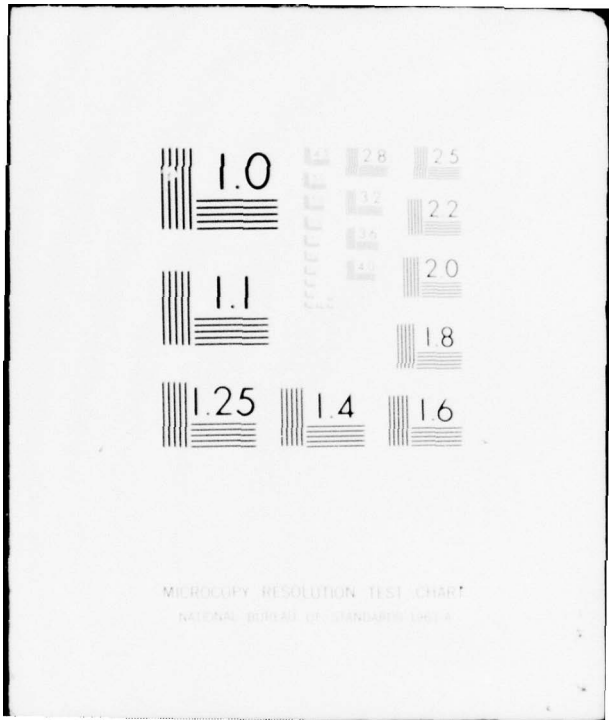
AD-A043 922 DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC F/G 9/2
MAINTENANCE MANUAL FOR AUDIT, A SYSTEM FOR ANALYZING SESCOMP SO--ETC(U)
AUG 77 R J WYBRANIEC, R REGEN

UNCLASSIFIED DTNSRDC-77-0075-VOL-1 NL

1 OF 2
AD
A043922


1.0
1.1
1.25
1.4
1.6
2.8
2.5
3.2
2.2
3.6
2.0
4.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

Report 77-0075

DAVID W. TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CENTER

Bethesda, Md. 20084

12

NW

MAINTENANCE MANUAL FOR AUDIT, A SYSTEM FOR ANALYZING SESCOMP SOFTWARE VOLUME 1

by

Robert J. Wybraniec
Richard Regen

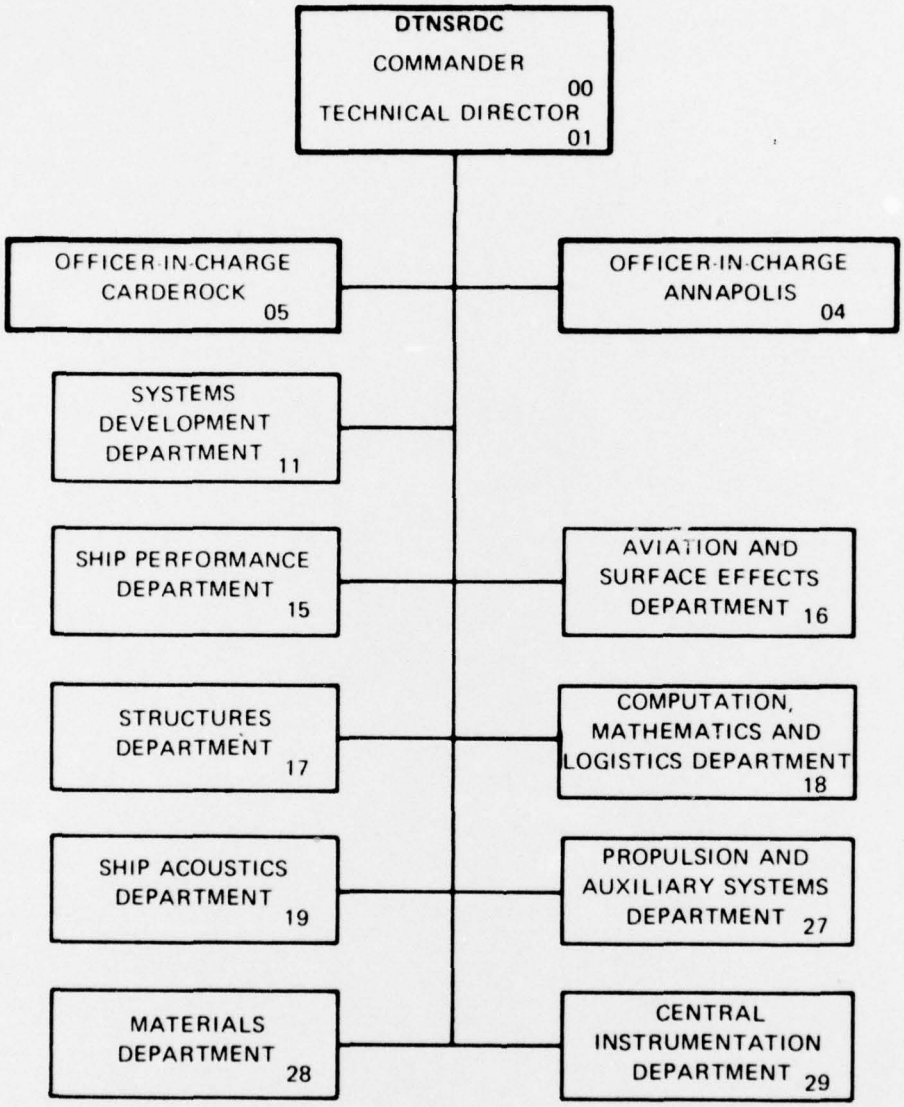
APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

COMPUTATION, MATHEMATICS, AND LOGISTICS DEPARTMENT
RESEARCH AND DEVELOPMENT REPORT

August 1977

Report 77-0075

MAJOR DTNSRDC ORGANIZATIONAL COMPONENTS



GPO 905-678

NDW-DTNSRDC 3960/43b (Rev. 11-75)
GPO 928-108

DTNSRDC ISSUES THREE TYPES OF REPORTS

(1) DTNSRDC REPORTS, A FORMAL SERIES PUBLISHING INFORMATION OF PERMANENT TECHNICAL VALUE, DESIGNATED BY A SERIAL REPORT NUMBER.

(2) DEPARTMENTAL REPORTS, A SEMIFORMAL SERIES, RECORDING INFORMATION OF A PRELIMINARY OR TEMPORARY NATURE, OR OF LIMITED INTEREST OR SIGNIFICANCE, CARRYING A DEPARTMENTAL ALPHANUMERIC IDENTIFICATION.

(3) TECHNICAL MEMORANDA, AN INFORMAL SERIES, USUALLY INTERNAL WORKING PAPERS OR DIRECT REPORTS TO SPONSORS, NUMBERED AS TM SERIES REPORTS; NOT FOR GENERAL DISTRIBUTION.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

(14) 1. REPORT NUMBER: DTNSRDC-77-0075-Vol-1

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

(6) 4. TITLE (and Subtitle): MAINTENANCE MANUAL FOR AUDIT, A SYSTEM FOR ANALYZING SESCOMP SOFTWARE, VOLUME 1.

(9) 5. TYPE OF REPORT & PERIOD COVERED: Final rept.

6. PERFORMING ORG. REPORT NUMBER

(10) 7. AUTHOR(s): Robert J. Wybraniec, Richard Regen

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS
David W. Taylor Naval Ship Research and Development Center
Bethesda, Maryland 20084

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
(See reverse side)

11. CONTROLLING OFFICE NAME AND ADDRESS
Navy Surface Effect Ships Project (PMS 304)
P.O. Box 34401 - Bethesda, Maryland 20084

(11) 12. REPORT DATE: August 1977

13. NUMBER OF PAGES: 184

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

(12) 185 p.

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

(16) SSH15, S0308

18. SUPPLEMENTARY NOTES

(17) SSH15001, S0308001

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

SESCOMP, SESCOMPSPEC's, Software Verification, Software Engineering, Reliability, Graph Theory, FORTRAN Software, Modules, Flow Analysis, Variable Precision Execution, Parser, Roll Call, Portability

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document provides the maintenance programmer personnel with the information to effectively maintain and use the AUDIT software. The AUDIT software examines FORTRAN computer programs or modules developed under the SESCOMP system for compliance with certain prescribed standards (SESCOMPSPEC's) and produces reports detailing the deviations from those standards. The AUDIT software also examines a program unit to detect and

(Continued on reverse side)

(Block 10)

63534N, 19588,
SSH15001 and S0308001,
11837001

(Block 20 continued)

report improper use of undefined variables along the program unit's possible paths. In addition, AUDIT has an option which enables the user to test the effect of changes in word length on the output of computer programs.

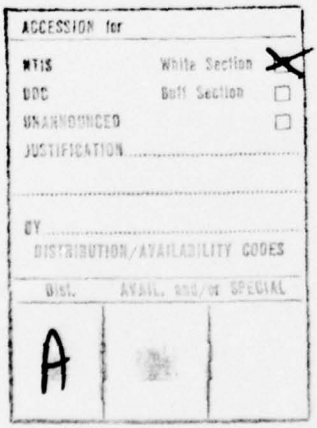
This document describes the entire AUDIT software, explains its applications, and gives a detailed description on how to use the software.

FOREWORD

The use and maintenance of AUDIT, a software system for analyzing SESCOMP contractor-supplied software, is documented as a set of four separately bound David W. Taylor Naval Ship Research and Development Center volumes sharing the common report number--DTNSRDC 77-0075:

- Maintenance Manual for AUDIT, a System for Analyzing SESCOMP Software, Volume 1
- Maintenance Manual for AUDIT, a System for Analyzing SESCOMP Software, Volume 2; Appendix B - Listings of the AUDIT Software for the CDC 6000
- Maintenance Manual for AUDIT, a System for Analyzing SESCOMP Software, Volume 3; Appendix C - Listings of the AUDIT Software for the UNIVAC 1108
- Maintenance Manual for AUDIT, a System for Analyzing SESCOMP Software, Volume 4; Appendix D - Listings of the AUDIT Software for the IBM 360

Volume 1 describes AUDIT and the use and maintenance of the AUDIT software. The other three volumes offer software listings for the CDC 6000, UNIVAC 1108, and IBM 360.



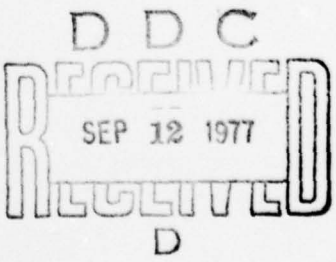


TABLE OF CONTENTS
Volume 1

LIST OF FIGURES

SECTION 1. GENERAL DESCRIPTION

1.1 Purpose of the Program Maintenance Manual. The objective for writing this Program Maintenance Manual for the AUDIT system is to provide the maintenance programmer personnel with the information necessary to effectively maintain the system.

1.2 System Application. The AUDIT system examines FORTRAN computer programs or modules developed under the SESCOMP system for compliance with certain prescribed standards (as set forth in the SESCOMPSPEC's[1 2 3 4 5]) and produces reports detailing the deviations from those standards. The AUDIT system also examines a program unit (main program or subprogram) to detect and report the existence of any undefined variables along the program unit's possible paths. AUDIT provides the user with an option which allows him to test the effect of changes in word length on the output of computer programs.

1.3 Equipment Environment. The AUDIT system is operational on three processors: CDC 6000 series, UNIVAC 1108, and IBM 360. AUDIT uses the following operating systems: CDC SCOPE 3.4, UNIVAC EXEC-8, and IBM OS-360.

AUDIT is available on tape for each of the three processors and on disk for the CDC 6600 at DTNSRDC. Card decks for each of the processors are also available. The mode of operation of the AUDIT system is controlled by card input. The programs, modules, or subprograms to be examined by AUDIT may be entered via cards, tape, or disk. The results of AUDIT which are of concern to the user are produced as printout.

1.4 Program Environment. The AUDIT system was designed to examine software developed under the SESCOMP system. SESCOMP, an acronym for Surface Effect Ship Computations, is the name of a system for procuring and managing modular computer programs developed for the Naval Sea System Command's Surface Effects Ship Project Office (PMS304). The background, objectives, and content of SESCOMP are described in Cuthbert, et al.[6] Although it is not required that the user of AUDIT have an understanding of the SESCOMP system, a basic knowledge of that system would help him to use AUDIT more effectively. AUDIT may be used to examine any FORTRAN software, not just the SESCOMP-produced software. The following attributes of the AUDIT system will aid users in producing more reliable software:

---

[1] References are listed on page 115.

(1) The flow analysis feature examines a program unit's paths to detect the use of undefined variables.
(2) The variable precision feature enables the user to test the effect of changes in word length on the output of a computer program.
(3) All aspects which affect portability among the four SESCOMP processors are examined.

The analytic capability of the AUDIT system combines with the SESCOMP programming standards to facilitate the production of more reliable software, as detailed in Culpepper.[7]

1.5 Conventions. Section 2 contains a detailed description of the AUDIT software and modes of operation. Section 3 contains a detailed description of the input and output conventions. Section 4 describes the operating procedure for all modes of operation of the AUDIT system. The listings of the AUDIT software for each of the three processors are each separately bound in Appendixes B, C, and D.

SECTION 2. SYSTEM DESCRIPTION

2.1 General Description. The Naval Sea System Command's Surface Effect Ship Project Office (PMS304) requested the Computation and Mathematics Department of the David W. Taylor Naval Ship Research and Development Center to develop computer software that would examine vendor-produced computer programs or modules for compliance with the SESCOMPSPEC's and would produce reports detailing the deviations. AUDIT, the computer software developed to carry out these functions, has four modes of operation. In the first, or auditing mode, AUDIT examines a program unit for conformance with the set of SESCOMPSPEC programming standards and issues a report detailing the deviations from those standards. In the second, or roll call mode, AUDIT examines a module for deviations from SESCOMP's roll call specifications (see Section 2.10 of SESCOMPSPEC5[5]). In the third, or variable precision mode, AUDIT tests the effect of changes in word length on the output of a computer program. In the fourth, or flow analysis mode, AUDIT examines a program unit to detect and report the existence of any undefined variables along the program unit's possible paths.

An important feature of AUDIT is the parser software, which checks expressions to see if they are syntactically and semantically correct. The parser software uses a syntax graph to define the legal grammar for expressions.

2.1.1 Audit Mode. The audit mode of AUDIT examines a program unit for adherence to the specifications in SESCOMPSPEC3[3], SESCOMPSPEC4[4], and SESCOMPSPEC5[5]. However, not all of the specifications are examined. Sections 2.1.1.1, 2.1.1.2, and 2.1.1.3 indicate which specifications are (or are not) checked by AUDIT.

2.1.1.1 SESCOMPSPEC3 Elements. The various sections of SESCOMPSPEC3 are considered in turn. AUDIT checks all of the SESCOMPSPEC3 specifications (and therefore the Standard) except as noted following. Comments are also made to amplify what is checked. The user himself must assume responsibility for checking the areas not covered by AUDIT.

2.1.1.1.1 Program Form. A diagnostic is issued when a non-FORTRAN character is encountered. All elements of the Standard concerning lines are enforced and appropriate diagnostics are issued for violations.

2.1.1.1.2 Data Types. All data types permitted by the Standard are recognized. Only characters of the FORTRAN character set are permitted as Hollerith types.

2.1.1.1.3 Data and Procedure Identification. The magnitudes of integer, real, double precision, and complex constants are not checked by the IBM 360 version of AUDIT. AUDIT does not check whether a dummy argument of an external procedure identifies a subroutine or external function not referenced in the program unit being analyzed. If a dummy argument of a program unit bears the same symbolic name as a subroutine or external function referenced within the program unit being analyzed, a diagnostic on the violation is issued.

2.1.1.1.4 Expressions. AUDIT does not check the requirement that no factor may be evaluated that requires raising a zero valued primary to a zero valued exponent.

2.1.1.1.5 Statements. AUDIT does not check the requirement that the integer variable i in an assigned GO TO of the form GO TO i, (k_1, k_2, ..., k_n) has been assigned one of the statement labels in the parenthesized list by an ASSIGN statement prior to the execution of the subject statement.

In a computed GO TO of the form GO TO (k_1, k_2, ..., k_n), i AUDIT does not ensure that i has been given a value j, $1 \leq j \leq n$, prior to execution.

In a DO statement of the form DO n i=m_1, m_2, (m_3), if m_1, m_2, and/or m_3 are variables, AUDIT does not check any of the following restrictions:

(1) At time of execution of the DO statement, m_1, m_2, m_3 must be greater than zero.
(2) The terminal parameter (m_2) must be no less than the initial parameter (m_1).
(3) The sum of the terminal parameter and the incrementation parameter minus one may not exceed 2**17-2.

The above three restrictions are checked if m_1, m_2, and/or m_3 are integer constants.

AUDIT does not check whether an array element name contains a variable subscript that, during execution of the program unit, assumes a value less than one or larger than the maximum length specified in the array declarator. For array element names having an integer constant subscript, AUDIT does not check for a zero subscript but does check for a subscript that is less than zero. AUDIT does check an integer constant subscript for exceeding the maximum length but does this only for non-executable statements (DATA and EQUIVALENCE).

AUDIT does not check that a formatted READ or WRITE statement may not read or create a record of more than 120

characters. It does not check that the first character of a formatted record for printing may not be a +.

AUDIT does not check that a FORMAT statement that is used for both input and output must end with a field descriptor or group of field descriptors. If there is an input/output list for a formatted READ or WRITE, AUDIT does not check that at least one field descriptor other than nH or nX must exist.

2.1.1.1.6 Procedures and Subprograms. For a FUNCTION statement of the form t FUNCTION f (a_1, a_2, ..., a_n) or a SUBROUTINE statement of the form SUBROUTINE s (a_1, a_2, ..., a_n), AUDIT checks that the a's, the dummy arguments, are each either a variable name or an array name but does not check that a dummy argument is not an external procedure name not referenced in the program unit being analyzed. If a dummy argument of a program unit bears the same symbolic name as a subroutine or external function referenced in the program unit being analyzed, a diagnostic on the violation is issued.

AUDIT checks the dummy arguments of FUNCTION and SUBROUTINE statements and the actual arguments used for referencing external functions and subroutines to insure that the arguments agree in order, number, type, and dimensionality with the argument definition in the SESCOMP Interface Definition file (see Section 3.2.1.3).

For a function subprogram AUDIT does not check whether the subprogram contains a statement that directly or indirectly references the function being defined.

The following is not checked: If a function reference causes a dummy argument in the referenced function to become associated with another dummy argument in the same function or with an entity in common, a definition of either within the function is prohibited.

2.1.1.1.7 Programs. The normal execution sequence is enforced by the flow analysis mode of AUDIT (see Section 2.1.4).

2.1.1.1.8 Intra- and Inter-Program Relationships. In the flow analysis mode, AUDIT traces the definition and redefinition of each variable in a program unit through all the possible program paths.

2.1.1.2 SESCOMPSPEC4 Elements. The following SESCOMPSPEC4 requirements are checked by AUDIT.

2.1.1.2.1 Usage. The symbolic name of the subprogram being analyzed is checked to insure that the name is in the Interface Definition file (see Section 3.2.1.3). The subprogram dummy arguments (if any) are checked to see if they agree in order,

number, type, and dimensionality with the subprogram's argument list as defined in the Interface Definition file. If the subprogram is a function subprogram, the type is checked with the Interface Definition file.

2.1.1.2.2 Labeled COMMON Storage. The symbolic name of each labeled COMMON block is checked to insure that the name is in the Interface Definition file.

2.1.1.2.3 Category 1 COMMON Storage. The size and type of each variable of a Category 1 labeled COMMON block is checked with the Interface Definition file.

2.1.1.2.4 Category 2 COMMON Storage. AUDIT checks that, for each Category 2 labeled COMMON block, the elements are grouped by type. AUDIT also checks that a module has at least one Category 2 block.

2.1.1.2.5 Blank COMMON Storage. If a module has blank COMMON storage, AUDIT checks that the size of the blank COMMON storage agrees with the size specified in the Interface Definition file.

2.1.1.3 SESCOMPSPEC5 Elements. The following SESCOMPSPEC5 requirements are checked by AUDIT.

2.1.1.3.1 BLOCK DATA. For a BLOCK DATA subprogram, the prescribed order of data types in a labeled COMMON block are checked.

2.1.1.3.2 COMMON Storage. The Category 1 labeled COMMON block SESCOM is checked to insure that the symbolic names assigned to each element of the block are the prescribed names. Each program unit is checked to see if it contains the mandatory Category 1 block SESCOM. In this case AUDIT does not distinguish whether the program unit being checked is a main program, module root program unit, ancillary subprogram, or extraordinary subroutine. The user should disregard the mandatory SESCOM diagnostic message if an ancillary subprogram or extraordinary subroutine START does not contain the SESCOM block.

For Category 2 and 3 labeled COMMON blocks, AUDIT checks that each and every element of each block is used by the subprogram. For modules with ancillary program units a misleading message may be issued. Since the audit mode analyzes only a single program unit, there is no way of knowing if an element in a Category 2 or 3 block is used in another program unit of the module. AUDIT also checks that a module has at least one Category 2 labeled COMMON block.

If a module has blank COMMON storage, AUDIT checks that the size of blank COMMON storage agrees with the size specified in the Interface Definition file.

2.1.1.3.3 Ancillary Subprograms. The symbolic name of each ancillary subprogram is checked to insure that the name is in the Interface Definition file (see Section 3.2.1.3).

2.1.1.3.4 Roll Call. AUDIT's roll call mode performs an analysis of a module's roll call actions (see Section 2.1.2).

2.1.1.3.5 Order of Statements. The prescribed order of statements is checked.

2.1.2 Roll Call Mode. AUDIT contains two options for checking the roll call action of the root program unit of a module. See Section 2.10 of SESCOMPSPEC5[5] for an explanation of a module's roll call actions.

(1) Option 1: AUDIT simulates the modules root program unit execution by executing the program unit for mode index values of 0, -1, ..., -11, -12 (a use count index of 1 is assumed). The output for each mode index is printed. The mode index value and the output device (X, Y, or Z) is designated for each output of a specific mode index value. For each mode index value, AUDIT also checks that each module referenced by the root program unit is referenced in the same roll-call mode.

(2) Option 2: For each mode index value (0, -1, ..., -11, -12), AUDIT checks that each module referenced by the root program unit is referenced in the same roll-call mode.

If an ancillary subprogram of a module references a module (A) not referenced by the root program unit of the module (for computational mode) and this module (A) is not referenced in the roll-call mode of the root program unit, AUDIT does not detect this deviation from the roll call specifications. The roll call mode uses subroutines CMPARE, MODID, and ROLCHK, and generates main program ROLCAL.

The roll call mode should only be used to examine the root program unit of a SESCOMP module.

2.1.3 Variable Precision Mode. If a contractor develops a program on a computer with a word length of 60 bits, it is important to determine whether it will produce correct results on a computer with a smaller word length. The AUDIT system provides a mechanism to assist in this determination. The variable precision mode allows the user to specify one of a group of truncation operators to be applied after each arithmetic operation. These truncation operators are in the form of variable precision function subprograms: Q1COMP, Q1DPRE, and Q1REAL (see Section 2.2.68). There are sets of truncation operators for the results of operations on real, complex, and

double precision numbers. The user selects the appropriate operator from each set to provide the desired degree of precision. The truncation operators simulate word lengths of 30 to 40 bits, 24 to 31 bits, and 30 to 35 bits for the CDC 6000, IBM 360, and UNIVAC 1108, respectively. The reference to the truncation operator following each arithmetic operation is inserted by AUDIT in the modified statements. Only two types of statements are modified, assignment statements and CALL statements.

An assignment statement is of the form V=E where V is a variable name or array element name and E is an arithmetic expression. AUDIT examines the arithmetic expression from right to left for triples of the form SAE_1 OP SAE_2 where OP may be +, -, *, or / and SAE is any valid FORTRAN expression. Each triple is rewritten in the form F_i (SAE_1 OP SAE_2), where F_i is Q1REAL, Q1COMP, or Q1DPRE, depending on whether the result of SAE_1 OP SAE_2 is real, complex, or double precision, respectively. The following example illustrates the method:

```
Given:   Y=A*B + C*D
Result:  Y=Q1REAL(Q1REAL(A*B) + Q1REAL(C*D))
```

A CALL statement is of the form CALL PROG (A_1, A_2, ..., A_n) where A_i are actual arguments. AUDIT examines each actual argument and generates a new CALL statement as follows:

(1) Constants, variable names, array element names, and array names are copied unchanged.
(2) FORTRAN expressions are processed as for assignment statements.

The following example illustrates the method:

```
Given:   CALL PROG (A, A(1,2), B, C+D)
Result:  CALL PROG (A, A(1,2), B, Q1REAL (C+D))
```

where A is an array and B, C, and D are real variables.

The variable precision technique is not intended to simulate the arithmetic peculiarities of any particular computer. Its only purpose is to permit the precision of arithmetic operations on a given computer to be varied. With this restriction in mind, the technique as implemented in AUDIT has one limitation:

- The calculations of FORTRAN library functions and of any subprogram not processed by AUDIT are not subjected to truncation.

2.1.4 Flow Analysis Mode. The flow analysis mode of AUDIT detects violations of ANSI FORTRAN[8] which are concerned with

inter- and intra-program relationships. In the design of the AUDIT system particular attention was paid to that section of the ANSI Standard which deals with the definition and "undefinition" of variables. This section states, for example, that if two variables of different types become associated by appearing in an EQUIVALENCE statement and one of the variables becomes defined, the other becomes undefined and may not be used in computation until redefined. The flow analysis is performed using some results obtained from the theory of compiler optimization provided in Allen[9] and Kennedy[10]. During the interstatement analysis of a program unit, AUDIT constructs the usual tables containing information about the variable names and their associations through COMMON and EQUIVALENCE statements. In addition a set of basic blocks is constructed. A basic block starts with:

(1) the first executable statement of the program unit, or
(2) a labeled executable statement, or
(3) the first executable statement after a RETURN, STOP, GO TO, or logical IF containing one of these forms, or
(4) the first statement after a DO or DO terminal statement.

A basic block ends with:

(1) any GO TO, STOP, RETURN, or logical IF containing one of these forms, or
(2) a DO or DO terminal statement.

A control flow graph is constructed using basic blocks as nodes and the control flow paths as edges. In the implementation of AUDIT, the information retained in the block consists of the names of the variables defined or undefined by the basic block and the names of the variables which must be defined prior to execution of the block. The symbol table is used to record the status of each variable, e.g., whether it is defined, undefined, an induction variable, or an ASSIGNed variable. Each path in the graph is traced through to detect and report improper use of undefined variables. A depth-first graph tracing algorithm is employed. In order to detect infinite loops and to facilitate diagnostic messages, a list of the basic block names in the path currently being traversed is maintained. As each new name is added to the list, the number of prior occurrences of that name is determined. When the count exceeds two, tracing of that path terminates and tracing of the next path begins. This heuristic seems satisfactory for the sort of programs normally produced by engineers. In a fairly complex program, however, it is possible that some legitimate paths may be missed. DO loops are traced only once. The flow graph is also analyzed to

insure proper nesting of DO loops and to detect illegal branches within DO loops. Subprogram references are checked to insure that the actual parameters are defined before the reference (for input parameters) and after the reference (for output parameters).

The flow analysis mode has two options: a full flow analysis and a moderate flow analysis. The full flow analysis allows a path to have a statement label that appears at most twice. The moderate flow analysis does not check any path in which a statement label appears more than once. Given the following coding:

```
      I=0
      J=0
    5 I=I+1
      CALL SUB(I)
      IF(I.LT.10) GO TO 5
   10 J=J+1
      CALL SUBB(J)
      IF(J.LT.10) GO TO 10
      RETURN
      END
```

the moderate flow analysis would check only one path: (5,10). The full flow analysis would check four paths: (5,10), (5,5,10), (5,5,10,10), and (5,10,10).

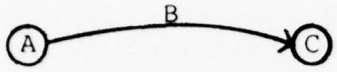
The moderate flow analysis will usually check less paths than the full flow analysis. The full flow analysis should be satisfactory for most program units. However, the moderate flow analysis is less time-consuming and should be used if the full flow analysis is taking too much time.

2.1.5 Parser Software. Arithmetic expressions, logical expressions, and I/O lists are parsed by AUDIT. Parsing involves determining whether an expression is syntactically and semantically correct. The software used to parse expressions is called the parser.

Part of the AUDIT parser is written in GIRL (Graph Information Retrieval Language), which is a high-level language developed at DTNSRDC and described in Berkowitz[11]. The GIRL-coded statements are translated into FORTRAN code by running the GIRL statements through the GIRL preprocessor. This FORTRAN code thus generated (and six other FORTRAN subroutines) is the parser software for AUDIT.

The parser requires a syntax graph as input. This graph is a plex data structure which completely defines all valid arithmetic and logical expressions, as well as I/O lists.

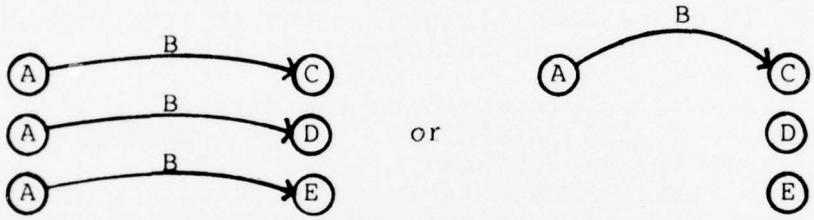
The parser can conveniently manipulate graph structures, i.e., insert, identify, retrieve, delete, and compare node-link-node triples such as the one shown following:



(single-value list)

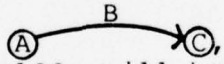
where A is the source node
B is the link
and C is the sink node.

Source nodes and links are random numbers generated by SUBROUTINE LVGRN. Sink nodes may be random numbers, integers, or Hollerith data. The triple can also be the component of a multi-value list represented in either of two ways.



The syntax graph is composed of single-value lists and multi-value lists linked together. The triples themselves are stored in a structure called GIRS (Graph Information Retrieval System) memory according to the following algorithm:

LOC = (A+B) modula MEMSZE.

For example, the triple 

, where A=52, B=73, and MEMSZE (GIRS memory size)=100, will be stored in GIRS location 25. LOC=(52+73) mod 100 = 125 mod 100 = 25.

The GIRS memory is constructed in the following way. Each single-value list requires four locations; each multi-value list requires 4(n+1) locations (where $n \geq 2$ is the number of sink nodes). The GIRS memory is stored in COMMON blocks LVVTR1, LVVTR2, LVVTR3, and LVVTR4.

The syntax graph is composed of nodes and links. Each node is a state in the graph. The parsing of an expression proceeds from state to state by way of the links. Each link is either a terminal symbol or STOP, ASSOC, or LEVEL. A terminal symbol is one of the following: PLUS, MINUS, SLASH, LPAR, RPAR, COMMA, STAR, EXP, LT, LE, GT, GE, EQ, NE, OR, AND, NOT, EQUALS, OPRAND. A constant or a variable is designated OPRAND. Each node and link is identified by a unique pseudo-random number.

Since FORTRAN is a finite state language, each state can lead to more than one state. Therefore it is necessary

to remember all the nodes that were visited in case it becomes necessary to back up to another point in the graph. This is accomplished through associates and levels. If the graph is in a certain state, the parser looks at the next terminal symbol in the input string. If that terminal symbol is a link from that node to another node, that link is selected. If not, an associate from that node is selected. If there is no associate, the level is selected from the last previous associate.

The syntax graph has three parts: (1) a FORTRAN expression graph for arithmetic expressions, (2) a Boolean expression graph for logical expressions, and (3) an I/O list graph for I/O lists. These three parts are shown in Figures 2-01, 2-02, and 2-03. An example to illustrate the parsing of an arithmetic expression follows. Consult the FORTRAN expression graph for help in following the logic.

```
The expression A+B*C is converted to the string
          OPRAND    PLUS    OPRAND    STAR    OPRAND.
This string is traced as follows:
     29    93    133    208    452
At this point it becomes necessary to back up to the last
previous associate and take the level
    133   235
back up to
     93   157
back up to
     29   519    plus sign found
     62    93    133    208    452
back up to
    133   235
back up to
     93   157    star found
    133   208    452    string empty
stop state found.
```

The parser software consists of the following subroutines: FORM,LVDLET,LVEXIT,LVFECH,LVFIND,LVGRN,LVNSRT, LVSETP,PARSE,PHONEY,PRNTS,RECOG,RECOV,SEMANT,SLEVEL, and SSTOP. LVDLET,LVFECH,LVFIND,LVGRN,LVNSRT, and LVSETP are part of the GIRS memory scheme. The remaining parser subroutines are originally written in GIRL and processed into FORTRAN. The explanation of the parser in this section should not be considered a comprehensive description. It is meant as a general description so as to aid in the description of the subroutines and COMMON blocks which follows. A more comprehensive description of the parser software is now under preparation at the Center in the Computer Sciences Division.

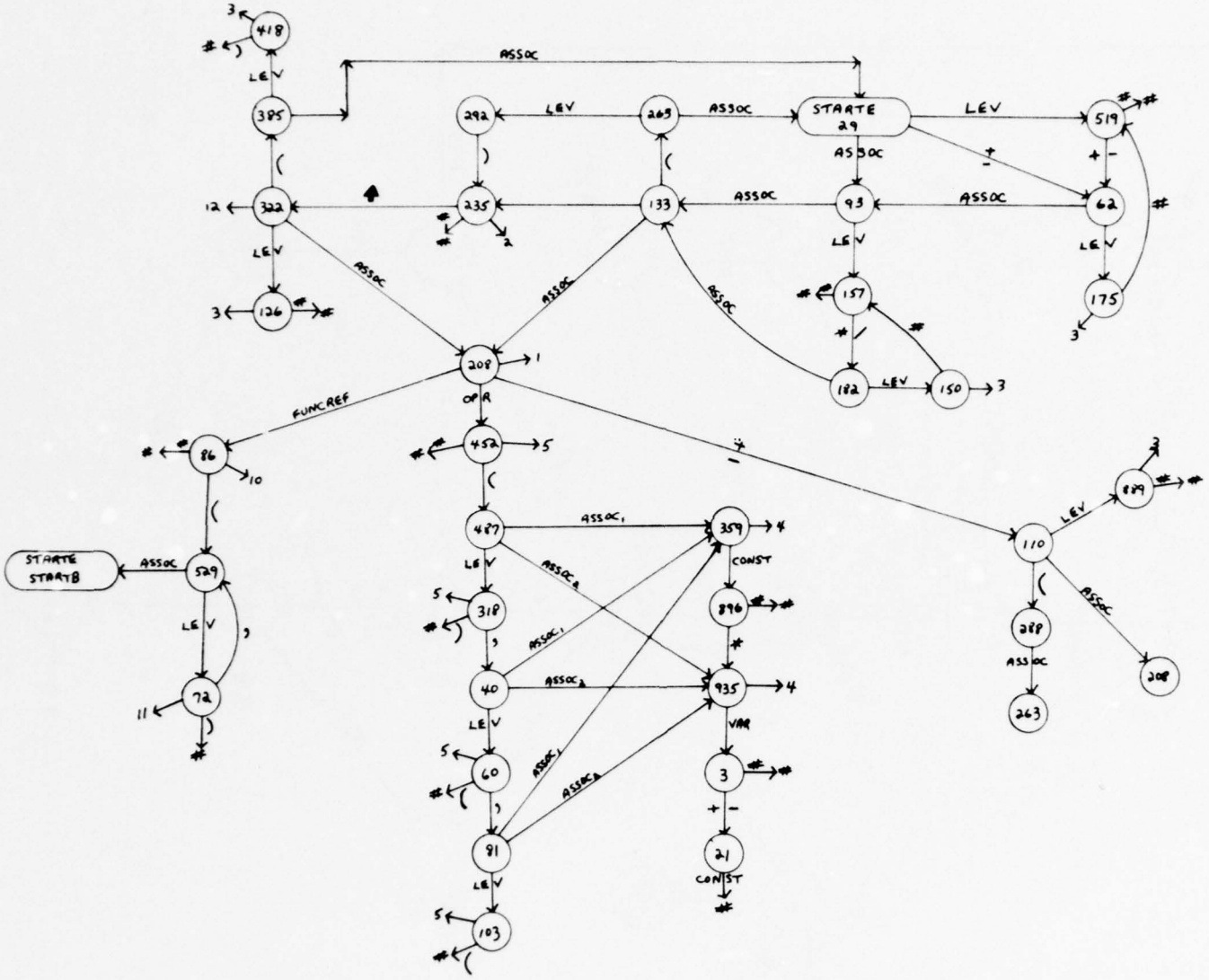


FIGURE 2-01. Arithmetic Expression Graph

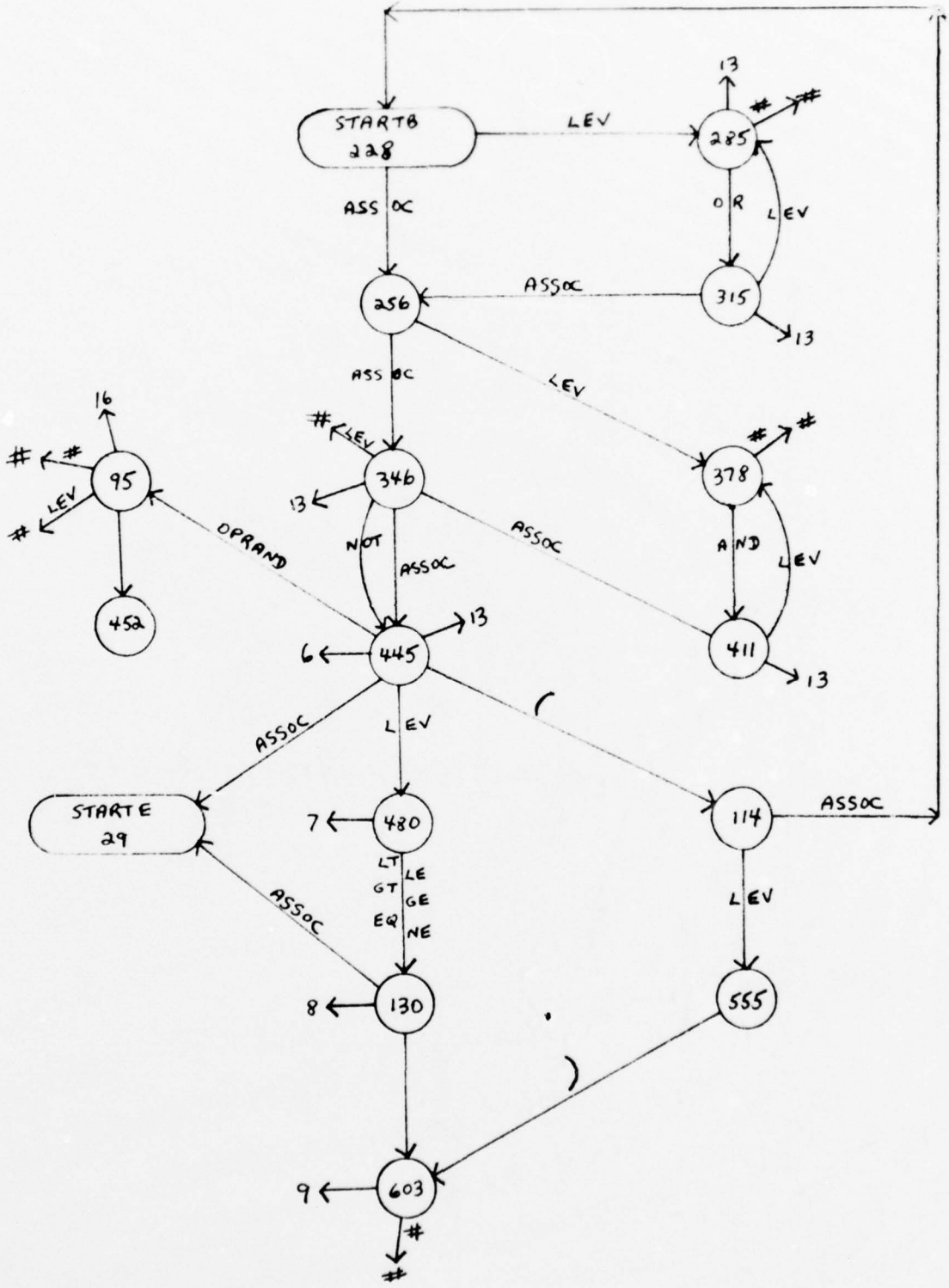


FIGURE 2-02. Logical Expression Graph

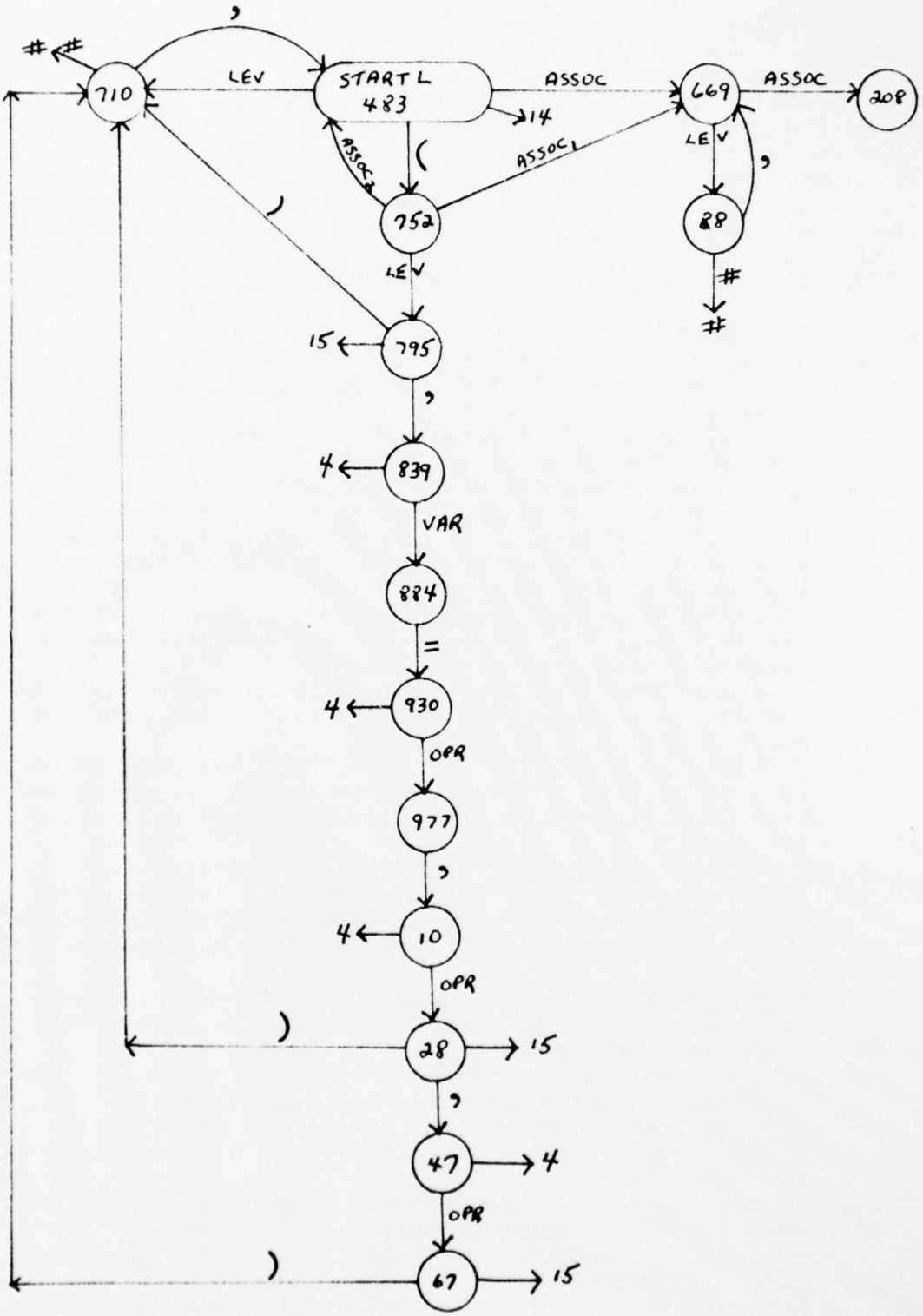


FIGURE 2-03. I/O List Graph

2.2 Detailed Description. All program units of the AUDIT software are described in Sections 2.2.1 through 2.2.89. After the main program the program units are listed in alphabetical order. The overall function(s) of each subprogram and clarifying remarks are described in the opening paragraph for each subprogram section. Following this opening paragraph, most subprogram sections also contain the logical flow through the program unit listed by numbered steps. All data elements in COMMON are described in Sections 2.2.90 through 2.2.122. Tables, arrays, and data elements used by the various program units are included.

2.2.1 Main Program.

1. Read the options card which contains the mode(s) of operation, input device, and the intrinsic function parameter.
2. Initialize global variables and tables.
3. Read the Interface Definition file.
4. Initialize non-global variables and tables.
5. CALL BUILD to assemble the next source statement.
6. Print the statement. If the statement is a comment or a blank, go to Step 5. If not, increment the statement counter and CALL CLASS to classify the statement.
7. CALL STATNO to process the statement number, if any.
8. Set character pointer to seven and save classes of current and preceding statements.
9. If it is the first statement of the program unit, test that it is a SUBROUTINE, FUNCTION, PROGRAM (for CDC only) or BLOCK DATA statement.
10. If it is not the first statement and the program unit is a BLOCK DATA subprogram, check that the statement is an EQUIVALENCE, DATA, DIMENSION, COMMON, or type statement.
11. Go to the appropriate statement processor.
12. Check all specification statements for proper order.
    If the statement is a FUNCTION statement, reposition the character pointer (if FUNCTION is preceded by a type).
    If the statement is an assignment statement, change its class (if it is a FUNCTION defining statement).
13. Write out the statement.
14. If it is the first statement of the program unit and the variable precision mode has been selected, generate type statements for Q1COMP and Q1DPRE (auxiliary variable precision subprograms to be loaded later) on logical unit 8.
15. If END statement not encountered go back to Step 5.
16. CALL SUBCHK to check the arguments (if any) of the program unit against the Interface Definition file.
17. CALL SYMTAB to display the symbol table.
    CALL GRT to update the global reference table.

CALL COMCHK to check COMMON blocks against the Interface Definition file.
CALL LOOPCK to check for proper nesting of DO loops and proper nesting of control throughout program unit.
CALL FLOWCK to perform flow analysis, if desired.

18. If there are more program units to be processed, go to Step 4.
19. CALL GLOTAB to display the global reference table.
20. If roll call mode has been selected, CALL GENROL to generate the main roll call program.
21. Rewind output devices.

2.2.2 SUBROUTINE ARIF. This subroutine processes arithmetic IF statements.

1. Verify the presence of the keyword IF and the presence of a left parenthesis.
2. CALL EXPR to code and analyze the expression.
3. CALL PARSE to parse the expression.
4. CALL FNCSTR to verify function references.
5. CALL BLKSTR to store basic blocks.
6. Set NBRNCH, the counter for branches out of the basic block, to zero.
7. Fetch the three statement numbers following the expression, store them in the statement number table, label them as having been referenced, and count the number of distinct branches out of the block.

2.2.3 SUBROUTINE ASGOTO. This subroutine processes assigned GO TO statements.

1. Check keyword spelling.
2. Fetch the GO TO variable, test that it is a simple integer variable, and store it in the basic block table.
3. Initialize the branch counter.
4. Fetch the next statement number in the list, store it in the statement number table, and flag it as having been referenced.
5. Check for duplicates. If the current statement number is not a duplicate, store it in the basic block table and increment the branch counter. Process the rest of the list and set the new block flag to 1.

2.2.4 SUBROUTINE ASSIGN. This subroutine processes ASSIGN statements.

1. Check keyword spelling.

2. Locate the statement number and flag it as having been referenced.
3. Check for the presence of the keyword TO.
4. Fetch the assigned variable and check that it is a simple integer variable.
5. Enter the index to the variable in the basic block table.

2.2.5 SUBROUTINE AUXIO. This subroutine processes REWIND, END FILE, and BACKSPACE commands for syntax errors.

2.2.6 INTEGER FUNCTION BITGET. This function retrieves a field of bits from a given position within the word ILOC. IPOS is the rightmost bit position (in ILOC) from which the field will be retrieved. IWIDTH is the number of bits in the field.

2.2.7 INTEGER FUNCTION BITPUT. This function stores a field of bits into a given position within the word ILOC. Bits are numbered left to right starting with 1. IVAL, right-justified and zero-filled, is the word which contains the field to be stored. IPOS is the rightmost bit position (in ILOC) into which the field will be stored.

2.2.8 SUBROUTINE BLKSTR. This subroutine is called after an expression has been parsed. It stores information in the basic block table.

1. Fetch next variable in the expression.
2. If variable is not associated with a function reference, store it as referenced and go to Step 1.
3. If variable is associated with a function reference, fetch the symbol table location of the function, fetch the Interface Definition file location of the function, and determine which of the function arguments is associatd with this variable.
4. If the variable associated with a function reference is a statement function, store the variable as referenced and go to Step 1.
5. Fetch the I/O status (IOSTAT) of this variable from the Interface Definition file. If IOSTAT=0, the variable is output and is stored as defined. If IOSTAT=1, the variable is both input and output, and is stored as both referenced and defined. If IOSTAT=2, the variable is input and is stored as referenced. If this variable appeared in an argument which was an expression, verify that the argument is not output (IOSTAT=0 or 1).

2.2.9 SUBROUTINE BUILD. This subroutine returns a character string of length N in array A. The character string consists of the next statement in the FORTRAN program unit being

processed. The flag IERR is set to 2 if an end-of-file condition has occurred. If there are more continuation cards than permitted, an error message is printed.

2.2.10 SUBROUTINE CAA. This subroutine packs a string of alphanumeric characters into a single word. The ISTR array contains the characters and MSTR is the number of characters. If MSTR is greater than 6, the string is too long. Each character is packed into ID in the leftmost available positions. After all the characters have been packed, ID is filled with blanks.

2.2.11 SUBROUTINE CAI. This subroutine converts a string of alphanumeric characters to an integer. The ISTR array contains the characters and MSTR is the number of characters. If MSTR is greater than 10, the integer is too large. Each character in the array is converted to its integer equivalent and multiplied by the proper power of ten. All these numbers are added up and stored in INTVAL. The size of INTVAL is checked against the maximum value (2**31-1).

2.2.12 SUBROUTINE CALL. This subroutine processes CALL statements.

1. Check keyword spelling.
2. Fetch the subroutine name and check it for validity.
3. Store the name of the called subroutine in the symbol table if it has not already been entered.
4. Perform the following actions if the subroutine doesn't have an argument list:
    a. If the subroutine is not in the Interface Definition file, store it there temporarily. Store this temporary location in the symbol table.
    b. If the subroutine is in the Interface Definition file, retrieve its location.
    c. Verify that the subroutine has no arguments.
5. Perform the following actions if the subroutine has an argument list:
    a. Parse the statement
    b. CALL FNCSTR to store function references.
    c. CALL BLKSTR to store basic blocks for flow analysis.
6. If the variable precision mode has been selected, CALL CNVRT to insert variable precision function references.
7. If the roll call mode has been selected and this subroutine is a subroutine module, CALL CALL2 to generate a call to ROLCHK.

2.2.13 SUBROUTINE CALL2. This subroutine is called by subroutines CALL (CALL statement processor) and INIT

(assignment statement processor) when the roll call mode has been selected and when the subprogram referenced in the CALL or assignment statement appears in the Interface Definition file as a function or subroutine module. Given a CALL statement of the form

CALL SESPL1 (N,L,M)

CALL2 transforms the statement into

CALL ROLCHK (1HS, 1HE, 1HS, 1HP, 1HL, 1H1).

The arguments of the call to ROLCHK are generated by taking the letters or numbers of the called subprogram one at a time.

2.2.14 SUBROUTINE CAR. This subroutine converts a string of alphanumeric characters to a real or double precision number. The RN array contains the characters and NCHAR is the number of characters. If a real number has more than 20 digits, or if a double precision number has more than 40 digits, processing is aborted. The characters are packed into array IHOL, right-justified and blank-filled. The DECODE statement converts the characters of the IHOL array to a real or a double precision number. The real or double precision number is then checked against the maximum and minimum values (1.0E+38 and 1.0E-38, respectively).

Note that this subroutine is not used in the IBM 360 or UNIVAC 1108 versions of AUDIT, since in those versions the magnitude of real, double precision, or complex constants is not checked.

2.2.15 SUBROUTINE CHKLST. This subroutine flags, as being initially defined, all those variables which have been declared in COMMON statements, DATA statements, or formal parameters (which are designated input), or which have been equivalenced to any of these. This subroutine is referenced by subroutine FLOWCK for the flow analysis mode.

2.2.16 SUBROUTINE CLASS. Given the character string A (which contains the statement currently being processed), this subroutine determines the type of statement. There are 36 possible types of statements and CLASS sets the variable ITYP to the appropriate value which indicates the type of statement. The codes for the 36 possible types of statements are given in Figure 2-04.

Statement Type	ITYP
Assignment	1
Assign	2
GO TO	3
Assigned GO TO	4
Computed GO TO	5
Arithmetic IF	6
CONTINUE	7
CALL	8
RETURN	9
STOP	10
READ	11
WRITE	12
REWIND	13
BACKSPACE	14
ENDFILE	15
Logical IF	16
DO	17
END	18
INTEGER	19
REAL	20
DOUBLE PRECISION	21
COMPLEX	22
LOGICAL	23
DIMENSION	24
COMMON	25
EQUIVALENCE	26
DATA	27
FORMAT	28
BLOCK DATA	29
SUBROUTINE	30
FUNCTION	31
PROGRAM	32
PAUSE	33
EXTERNAL	34
Statement function	35
Unclassified	36

FIGURE 2-04. Statement Type Codes

CLASS recognizes all statement functions initially as assignment statements. They are subsequently reclassified as statement functions by subroutine INIT. CLASS makes its classification by first testing to see whether the string is a valid assignment statement. If it is not, then the first few characters of the string are examined to determine which of the FORTRAN keywords are present. The appropriate value of ITYP is then set.

2.2.17 SUBROUTINE CMPARE. This subroutine is used in the roll call mode to verify that all of the modules referenced by the module being processed were referenced in the SESCOMP roll-call mode. CMPARE requires as input the AUDIT module list, which resides on logical unit 9, and the roll check module list, which resides on logical unit 3. CMPARE checks that all modules in the AUDIT module list appear on the roll check module list for all valid values (0 to -11) of the mode index.

2.2.18 SUBROUTINE CNVRT. This subroutine converts coded output from subroutine PARSE into BCD (Binary Coded Decimal) and returns it to array A. The PARSE output is the result of the variable precision transformation described in Section 2.1.3.

1. Copy the left side of an assignment statement to the output array.
2. If an arithmetic operator in the input string is found, copy out the appropriate operator.
3. If a logical operator in the input string is found, copy the appropriate operator to the output array.
4. If the codes for the variable precision functions Q1REAL, Q1COMP, and Q1DPRE are found, copy out the appropriate function name.
5. Process the code for a constant or variable name.
6. Fetch the length of the name.
7. Fetch the location of the name in the symbol table.
8. Deal with constants.

2.2.19 SUBROUTINE COM. This subroutine processes COMMON statements by checking the syntax of the statements and by entering the declarative information in the statement into the symbol table.

1. Check keyword spelling.
2. Add each COMMON block to the symbol table.
3. Chain together the variable names in each COMMON block as they are encountered. The COMMON block name points to the first variable in the block and the last variable in the block. The last variable in a chain contains a pointer to the beginning of the chain. Each variable in the chain points back to the COMMON block name.
4. Maintain counts of the number of COMMON blocks and their lengths.
5. Test each block name and each variable name to insure that it is in the proper class, i.e., that it is not also the name of a function.

2.2.20 SUBROUTINE COMCHK. This subroutine is referenced after all statements of the program unit being analyzed have

been processed. It checks the validity of all COMMON blocks that appeared in the program unit.

1. Get next COMMON block from the symbol table.
2. Fetch the category and size of COMMON block from the Interface Definition file and check the size (for Category 1). The sizes of Category 2 and 3 blocks are checked for consistency if they appear in other program units being analyzed.
3. Fetch next variable in COMMON block, along with its type and size (if it is an array).
4. If variable is complex or double precision, verify that it begins on an even word within the COMMON block.
5. If this is a BLOCK DATA subprogram, verify that the variables are in their prescribed order (real, integer, double precision, complex, and logical).
6. If this is a Category 2 COMMON block, verify that it is grouped by type.
7. If this is a Category 2 or 3 COMMON block, verify that all the variables are used.
8. If this is a Category 1 COMMON block, check the variable types against the Interface Definition file.
9. Check for the existence of COMMON block SESCOM and check for its prescribed variables.

2.2.21 SUBROUTINE COMEXT. This subroutine is referenced by subroutine EQUIV (EQUIVALENCE statement processor) to determine whether an EQUIVALENCE statement extends COMMON and whether a double precision or complex variable begins on an even location within COMMON.

1. Fetch the location in the symbol table of the COMMON block which contains the equivalenced variable.
2. Fetch the location of the last variable in the block.
3. Fetch the size of the block.
4. Calculate the total number of storage units which precede the equivalenced variable in its COMMON block.
5. Fetch the next variable and, if it is equivalenced, transfer out of the loop to statement number 25.
6. If the variable is dimensioned, go to statement number 10. Otherwise increment the storage unit counter and continue the loop.
7. If the variable is dimensioned, compute the number of storage units required from the dimensions of the array.
8. From the above result, calculate the number of storage units in the COMMON block which follow the equivalenced variable.
9. If the variable is double precision or complex, test that it begins on an even location.

10. Test whether COMMON is extended.
11. Fetch the offset (see SUBROUTINE EQUIV) of the equivalenced variable in the COMMON block.
12. Fetch the offset of each variable in the equivalence chain, calculate the first and last storage unit locations occupied by the variable, and check whether these locations lie within the storage area occupied by the COMMON block.

2.2.22 SUBROUTINE COMSCH. Given the COMMON block name stored in NXTID, this subroutine sets ISRCH(3) equal to 1 if the name is found in the symbol table and 0 if it is not.

2.2.23 SUBROUTINE CTGOTO. This subroutine processes computed GO TO statements.

1. Check keyword spelling and for the presence of a left parenthesis.
2. Reserve a space in the basic block table for the GO TO variable. JBLOCK is used to store the index to that space.
3. Initialize the branch counter.
4. Procure and check the statement numbers in the statement number table.
5. Set the flag in the statement number table which indicates that the statement number has been referenced.
6. If the statement number has occurred previously in this statement, go to statement number 17. Otherwise store the statement number in the basic block table and increment the branch counter.
7. After all the statement numbers have been processed, fetch the GO TO variable and test that it is a simple integer variable.
8. Store the index of the variable in the basic block table, set the new basic block flag to 1, and return to the referencing program.

2.2.24 SUBROUTINE DATA. This subroutine processes DATA statements.

1. Check keyword spelling and perform some initialization.
2. Fetch the next variable name and enter it in the symbol table if it is not already there.
3. Test the variable for a previous appearance in a DATA statement.
4. Test the variable to determine if it is a formal parameter.
5. Determine whether the variable is in COMMON. Issue an error message if the variable is in blank COMMON

or if the variable is in labeled COMMON and this is not a BLOCK DATA subprogram.

6. Test for a left parenthesis occurring next in the statement, in which case the variable name preceding the left parenthesis must be the name of an array.
7. If the variable is the name of an array, check the subscripts against the dimensions given in the DIMENSION statement for the variable in question.
8. If a dimensioned variable is used without a subscript, calculate the size of the array.
9. Begin processing the constant list. To detect any discrepancies between the number of variables in the variables list and the number of constants in the constant list, two counters (LST1SZ and LST2SZ) are used to keep the respective counts. Initialize the repeat counter.
10. Fetch the next element and, if it is a Hollerith string, go to statement number 47. If it is not a Hollerith string and if the element is not an integer, go to statement number 45.
11. The element is either an integer or a repeat count for a following element. If it is a repeat count, convert it to binary and store it in NRPEAT.
12. Check whether a sign precedes the constant.
13. Test the constant to see if it is real, integer, complex, Hollerith, or logical.
14. Increment the constant list size counter. The next element in the string being processed should be either a comma or a slash. If it is a comma, go to statement number 40 to get the next element. If it is a slash, compare the two list size counters and, if they are not equal, print an error message. If the element following the slash is a comma, go to statement number 8 to process another list. If it is not a comma, a blank signifies the end of the statement.

2.2.25 SUBROUTINE DESCRP. Given the starting point IDESST in the string A, this subroutine sets the variables IDES to 1 if the substring which follows is a valid format field descriptor and 0 if it is not. A valid format field descriptor is one of the following:

srFw.d
srEw.d
srGw.d
srDw.d
rIw
rLw
rAw
$nHh_1 h_2 \ldots h_n$
nX

where w and n are non-zero integer constants, d is an integer constant, r is an optional non-zero integer constant, and s is optional and represents a scale factor designator of the form mP, where m is an integer constant or minus followed by an integer constant.

2.2.26 SUBROUTINE DIMEN. This subroutine processes DIMENSION statements by checking keyword spelling and then adding to the symbol table the dimension information for each variable in the list. Errors diagnosed by this subroutine include exceeding allowable array bounds, illegal use of variable dimensions, illegal use of subprogram names, dimensioning previously dimensioned variables, and other illegal uses of dimensioned variables.

2.2.27 SUBROUTINE DO. This subroutine processes DO statements.

1. Check keyword spelling.
2. Fetch the statement number of the terminal statement, enter it in the statement number table, and flag it as having been referenced.
3. Store the terminal statement number index and the DO loop number in the DO stack.
4. Fetch the induction variable name, store it in the symbol table, and check that it is a simple integer variable.
5. Check for the presence of the equals sign.
6. Enter the induction variable name in the basic block table.
7. Store the induction variable name in the DO stack.
8. Make a detailed check of the three DO parameters defining the initial value, the final value, and the increment of the induction variable. A "998" flag is inserted in the basic block to show that control passes to the next basic block (see FUNCTION NXTBLK).

2.2.28 SUBROUTINE EQUIV. This subroutine processes EQUIVALENCE statements by checking them for proper syntax and by analyzing the declarative information in the statements for consistency and conformance with the Standard. Although the function performed by EQUIV is conceptually simple, the implementation is sufficiently complex to warrant the following detailed explanation.

In an EQUIVALENCE statement of the form EQUIVALENCE (A,B), (C,D), the variables within parentheses are said to be in equivalence groups. Simple (undimensioned) variables in an equivalence group occupy the same storage location. Within the symbol table, space is set aside to permit the linking together in a chain of all the variables in an equivalence group. An additional field in the symbol table entry, called

the offset field, is provided to be used when one or more of the variables in the equivalence chain is dimensioned. For example, if we had the statements DIMENSION B(10,10), C(10,10) and EQUIVALENCE (A,B(1), C(2)), then A and B would have an offset of zero and C would have an offset of -1 to indicate that the array C starts one location before the variable A and the array B. If the statement EQUIVALENCE (B(2),D) is included with the other two statements, then since B(1) has an offset of zero, B(2) will have an address of 1 relative to the rest of the equivalence group. Thus, D would have an offset of 1.

Errors diagnosed by EQUIV include:

a) Directly or indirectly causing two different elements of the same array to be equivalenced to each other.
b) Directly or indirectly introducing gaps in the storage map of the array.
c) Extending COMMON in either direction.

1. Check keyword spelling.
2. Test for the left parenthesis which starts an equivalence group.
3. Perform some initialization, including setting the base offset value to zero, and begin processing an equivalence group. CALL GNLE to get the next language element in the statement. Test the name to determine whether the language element was an identifier, and store the name in the symbol table, if it is not already present.
4. Process subscripted variables.
5. Compute the offset for the variable.
6. Process variables which have not previously appeared in an EQUIVALENCE statement. Set the "has been equivalenced" flag in the symbol table entry. Test to see if this is the first element in a chain. If not, store the location of this element in the symbol table entry of the previous element in the chain. In any case store the offset for this element in the symbol table entry for that element.
7. If the variable has previously appeared in an EQUIVALENCE statement, merge the two equivalence chains and recalculate the offsets of the merged chain.
8. Merge the chain and adjust the offsets in the merged chain.
9. Check whether two variables in COMMON have been equivalenced, whether COMMON has been extended, and whether syntax errors have been made in the EQUIVALENCE statement.

2.2.29 SUBROUTINE ERROR. This subroutine prints an appropriate error message when a program unit of the AUDIT software finds an error in the program unit being analyzed.

2.2.30 SUBROUTINE EXPR. Given the starting location of a character string representing a FORTRAN expression or an I/O list, this subroutine encodes the expression for later processing.

The character string is examined and output array STR is generated. Figure 2-05 shows the encoding for operators.

Character	Code
+	-1
-	-2
/	-3
(	-4
)	-5
,	-6
*	-7
**	-8
.LT.	-9
.LE.	-10
.GT.	-11
.GE.	-12
.EQ.	-13
.NE.	-14
.OR.	-15
.AND.	-16
.NOT.	-17
=	-18

FIGURE 2-05. Encoding Operator Codes

Variable names are encoded as

$$I+10^4 A+10^5 B+10^6 M$$

where I is the starting location of the variable name in the character string,

A = 0 if the variable is real
  = 1 if the variable is complex
  = 2 if the variable is double precision
  = 3 if the variable is integer
  = 4 if the variable is logical
B = 0 if the variable is simple
  = 1 if the variable has one subscript
  = 2 if the variable has two subscripts

= 3 if the variable has three subscripts
= 4 if the substring is a constant
= 5 if the name is a function reference

M is the length of the substring.

The number of entries in the encoded array is stored in NSTR.

1. Test for an end of the expression, which is one of the following:
   a. no more characters in string
   b. equals sign in an assignment statement
   c. right parenthesis in a logical IF statement.
2. If a terminal symbol is not encountered encode character as indicated in Figure 2-05.
3. If character is a name followed by a left parenthesis, determine whether it is a function reference or an array element and encode it accordingly. If a function is a basic external or intrinsic function, get its type from the Interface Definition file.
4. If function name is not followed by a left parenthesis, verify that the variable is the function name within the function-defining subprogram (see case B of SUBROUTINE INIT).

2.2.31 SUBROUTINE EXPRCK. After an assignment statement has been processed, this subroutine is called to see if the assignment is legal. An assignment statement is of the form V=E, where V is a variable name and E is an expression. Legal assignments are listed in Figure 2-06.

V	E
Integer	Integer
Integer	Real
Integer	Double Precision
Real	Integer
Real	Real
Real	Double Precision
Double Precision	Double Precision
Double Precision	Integer
Complex	Complex
Logical	Logical

FIGURE 2-06. Legal Assignment Rules

2.2.32 SUBROUTINE FLOWCK. This subroutine examines all possible paths through a given program unit to determine whether or not all the variables required for the computation at a given point in the subprogram are properly defined in the sense that SESCOMPSPEC3 requires. FLOWCK is used during the flow analysis mode.

The approach adopted is based on the work of Ramamoorthy[12]. The method used is to generate one by one all the complete paths in the program unit. After each path is generated, the flow of variable definition and undefinition is traced out along that path. A complete path is defined to be a path which commences at the beginning of the program unit, terminates in a STOP or RETURN statement, and in which no statement number appears more than twice. Strictly speaking, this definition excludes some possible paths from considerations but most practical programs are fully analyzed. Two data structures provide the information required for this analysis: IBLOCK and ISTACK, which are fully described in Sections 2.2.91 and 2.2.92, respectively.

Briefly, however, IBLOCK is an array of basic blocks, a basic block being a group of statements in the source program which physically appear together in the program, which are located in the same DO loop, and which are executed together (See Section 2.1.4). ISTACK is an array which, for each DO statement, contains the statement number of the terminal statement, the index in the symbol table of the induction variable, and the index in the ISTACK array of the DO in which this DO is nested. Some of the intermediate data structures used by FLOWCK are:

a. FLWLST - This list contains the indices of the basic blocks which are on the path currently being considered. If the index in FLWLST is negative, there remain some branches in the block which have not yet been examined.
b. ISTCK - This list contains the branch list for the basic block being processed. The last branch is flagged as being negative.

In addition, a flag in the symbol table is used to maintain the status of each variable, i.e., whether it is undefined, defined, an induction variable, or assigned.

1. If there are structural syntax errors in the program unit, do not perform the flow analysis and issue a message.
2. CALL CHKLST to flag, as being initially defined, all variables in COMMON and DATA statements, as well as all formal parameters which are designated as logical input. Also flag, as being initially defined, any variable which is equivalenced to any of the above.
3. Initialize symbol table for next path and initialize current block at 1.
4. Generate the next path (Steps 5-10).
5. If path is non-empty, count the number of occurrences of the current basic block in the path. If greater than the limit (2 for full flow analysis and 1 for moderate flow analysis), go to Step 23 and attempt to generate a new path.

6. If path is okay so far, add the current basic block to the path (FLWLST).
7. Get the next basic block which this block branches to and make it the current block.
8. If there is more than one branch from the block, set the block index to negative in FLWLST to indicate that there are more branches to consider.
9. Store the last branch in the block in ISTCK and flag it as negative to indicate that it is the last branch. Store all other branches as positive.
10. If it is not the end of the path, go to Step 5.
11. Path has been generated.
12. Increment path counter.
13. Get the next block in path and its statement label and DO loop.
14. Analyze all variables in the block. Get variable class from IBLOCK array (see COMMON block BASBLK).
15. If variable is defined, check that the variable is not a DO parameter, a DO index, or a variable dimension. If variable is equivalenced, set all equivalenced variables of the same type to defined and set all equivalenced variables of different type to undefined. Set status flag to 1 if variable is defined.
16. If variable is referenced, check that the variable is defined and not assigned.
17. If variable is a DO index, check that the variable is not a variable dimension and is not currently a DO index. Set status flag to 2 for DO index variable.
18. If variable is assigned, check that the variable is not a variable dimension nor a DO index. Set status flag to 3 for assigned variable.
19. If variable is referenced by an assigned GO TO, check that the variable is assigned.
20. If variable is made undefined, set status flag to 0.
21. If variable is a DO parameter, check that the variable is defined and not assigned. Set status flag to 4 for DO parameter variable.
22. If there are more variables, go to Step 14.
23. At this point path processing is complete. Path has either been analyzed or rejected.
24. Begin to generate the next path.
25. Starting at the bottom of the flow path, find the first negative index indicating a branch point. If none remain, go to Step 28.
26. Fetch the branch from the top of the stack. If this branch is negative, indicating that it is the last branch from this point, reset the flow path index to positive.
27. Reset the stack counter and current path index and go to Step 4 to complete generation of this path.

28. All paths have been processed. Check that each statement label appears in a path.

2.2.33 SUBROUTINE FNCSTR. This subroutine is called after parsing a CALL or assignment statement to check all function references against the Interface Definition file.

1. If there are no function references, RETURN.
2. Fetch next function reference from FNCLOC.
3. If it is a statement function, call STFNC to process the statement.
4. Fetch number of arguments and function type.
5. If function has previously appeared, go to Step 7.
6. Search the Interface Definition file for function name.
   a. If found, store file location in symbol table and go to Step 8.
   b. If not found, issue diagnostic and store name in the temporary Interface Definition file. Store file location in the symbol table. Store function type, number of arguments, argument type, dimensionality, and I/O status in temporary Interface Definition file.
7. Fetch Interface Definition file location of function.
8. Fetch number of arguments from Interface Definition file and check for correctness.
9. Compare type, dimensionality, and I/O status of each argument against the Interface Definition file.

2.2.34 SUBROUTINE FORM. This subroutine creates the format used to print the error diagnostic issued by SUBROUTINE PRNTS.

2.2.35 SUBROUTINE FORMEL. Given the character string D generated by SUBROUTINE GNLE, this subroutine processes language elements. Note that SUBROUTINE CAR is not referenced for the IBM 360 or UNIVAC 1108 versions of AUDIT.

1. CALL CAA to process identifiers.
2. Compute size of Hollerith string. Adjust the character pointer JPTR. Check validity of each character in the string.
3. CALL CAR to process real numbers (for CDC only).
4. CALL CAI to process integer numbers.
5. For complex numbers, CALL CAR to process both real numbers which comprise the complex number (for CDC only).
6. Do not process operators and logical constants.

2.2.36 SUBROUTINE FRMAT. Given a character string A, this subroutine determines whether or not it is a valid SESCOMP FORMAT statement, i.e., whether it has the form $(q_1 t_1 z_1 t_2 z_2 \ldots\ldots t_{n-1} z_{n-1} t_n q_2)$, where each q is a series of slashes or is

empty, each t is a field descriptor or group of field descriptors, each z is a field separator, and n may be zero. FRMAT sets IFRMT to 1 if the FORMAT statement is valid and 0 if it is not. FRMAT calls subroutines GROUP, SEPAR, and DESCRP to help process the FORMAT statement.

2.2.37 SUBROUTINE GENROL. When the roll call mode has been selected, this subroutine generates a main program which is then compiled by the FORTRAN compiler along with the program unit (should be a root program unit of a module) being analyzed. This main program then calls the module root program unit in the SESCOMP roll-call mode for all permissable (and one error value) values of the mode index. The basic form of the program generated by GENROL is as follows:

```
      PROGRAM ROLCAL (OUTPUT,TAPE6=OUTPUT,TAPE3,TAPE9,
     *TAPE10,TAPE11,TAPE12,TAPE13,TAPE14,TAPE15)
      COMMON/NAME0/IX0(N0)
      COMMON/NAME1/IX1(Ni)
      COMMON/NAME2/IX2(N2)
         .
         .
      COMMON/SESCOM/IXj(Nj)
         .
         .
      COMMON/NAMEn/IXn(Nn)
      J=1
      MODE=m
      DO 10 I=1,13
      J=J-1
      DO 1000 K=1,N0
      IX0(K)=1
 1000 CONTINUE
      DO 1001 K=1,N1
      IX1(K)=1
 1001 CONTINUE
         .
         .
      DO 100j K=1,Nj
      IXj(K)=1
 100j CONTINUE
      IXj(17)=10
      IXj(20)=11
      IXj(23)=12
         .
         .
      DO 100n K=1,Nn
      IXn(K)=1
 100n CONTINUE
      CALL SUBR(A1,A2,A3,....,An,J)
      IF(MODE.EQ.3) GO TO 5
      CALL MODID(J)
```

```
    5 ENDFILE 3
   10 CONTINUE
      CALL CMPARE
      REWIND 13
      REWIND 14
      REWIND 15
      STOP
      END
```

The following items pertain to ROLCAL:

- NAMEi is the name of the i^{th} COMMON block of the module root program unit.
- IXi is a dummy COMMON block element name
- Ni is the length of the i^{th} COMMON block.
- SESCOM is the mandatory labeled COMMON block (Nj should be equal to 25)
- m is one of two options for the roll call mode. If m=2 most roll call actions are performed by the program unit. If m=3 most roll call actions are performed by the SESCOMP roll call utility module ROLCOL.
- SUBR is the name of the program unit being analyzed.
- MODID is an auxiliary subroutine called to print out what was written on each output device by SUBR.
- CMPARE is an auxiliary subroutine called to verify that all of the modules referenced by SUBR were referenced in the SESCOMP roll call mode.
- The PROGRAM statement is only used for the CDC version of AUDIT. This statement is omitted for the IBM 360 and UNIVAC 1108 versions.

The logic for SUBROUTINE GENROL follows:

1. Perform initialization, including creation of COMMON block numbers.
2. Generate the PROGRAM statement (for CDC version only).
3. Generate the COMMON statements for each COMMON block.
4. Generate the top of the loop for stepping through the values of the mode index.
5. Generate the code to set each COMMON block element to 1.
6. Generate the code to define the SESCOM output devices.
7. Generate the call to the program unit being analyzed, making sure that the proper number of arguments are included.
8. Generate the remainder of the program.

2.2.38 SUBROUTINE GLOTAB. This subroutine displays the global reference table.

1. Display heading

2. Fetch external references from the external reference list. Fetch class of each external reference from the Interface Definition file and display all externals.
3. Fetch COMMON block names from the COMMON block list. Fetch size and class of each COMMON block from the Interface Definition file and display all COMMON blocks.
4. Fetch the name of each program unit being analyzed from the subprogram list. Fetch the class of each program unit from the Interface Definition file and display all program units.

2.2.39 SUBROUTINE GNLE. Given the character string A and the pointer JPTR into the string, this subroutine finds the next language element, stores it, and classifies it. The element is stored in array D, and M is set to its length. The pointer JPTR is set to the first location in A following the element. The variable JTYP is set according to Figure 2-07.

JTYP	Language Element
0	Blank
1	Arithmetic operator
2	Identifier
3	Hollerith string
4	Real number
5	Integer number
6	Complex number
7	Logical constant or operator or relational operator
8	Invalid

FIGURE 2-07. Language Element Codes

2.2.40 SUBROUTINE GOTO. This subroutine processes GO TO statements.

1. Check keyword spelling.
2. Check the statement number.
3. Flag the statement number in the statement number table as having been referenced.
4. Process the basic block by storing the statement number in the basic block, setting NBRNCH, the counter for the number of branches out of the block, to 1, and setting NB, the "new block" parameter, to 1.

2.2.41 SUBROUTINE GROUP. Given IGRST and IGRND, the beginning and end of a character string, this subroutine sets IGRP to 1 if that string is a valid group of format field descriptors and 0 if it is not.

2.2.42 SUBROUTINE GRT. This subroutine is called after each program unit has been processed. It maintains a composite list of names (global reference table) of subprograms and COMMON blocks referenced by each of the program units being analyzed.

1. Fetch the name of the first subprogram in the input table. If there are no more entries, go to Step 5 to process COMMON block names.
2. If the name is a statement function, bypass that name and get the next name.
3. Search the external reference table to see if the name is already stored. If not, store the Interface Definition file location of the name in the external reference table.
4. If operating under the roll call mode, and if the subprogram is a function or subroutine module, write the name on logical unit 9 to be used later during roll call check.
5. Process COMMON block information.
6. Fetch the first COMMON block name used by the program unit just processed; if there are no more COMMON blocks, return to the referencing program unit. If there is blank COMMON, bypass and fetch the next name.
7. Search the Interface Definition file for COMMON block name.
8. If the name is found, get the COMMON block category.
9. If category 1, go to step 12.
10. If not category 1, store the size and go to step 12.
11. If the name is not found, store the name and size in the temporary Interface Definition file.
12. Store the Interface Definition file location in the symbol table.
13. Search the COMMON block list for the name; if not found, enter the Interface Definition file location of the COMMON block in the COMMON block list.

2.2.43 FUNCTION ICOMP. Given an array IVAR of length 2 and a symbol table location, this subprogram compares IVAR against the first two columns in that symbol table location. ICOMP returns a value of 1 if there is a match and 0 if not.

Note that this subprogram is used only for the IBM 360 version of AUDIT since the IBM 360 has a four-character word. A variable may be as many as six characters long, so two words may be needed to store it. The CDC and UNIVAC 1108 version of AUDIT can store the variable in a single word.

2.2.44 SUBROUTINE IMPTYP. This subroutine first examines the "type set" flag. If the type has already been set, no typing occurs. If not, IMPTYP performs implicit typing by examining the initial character of a variable name to see if it is

I, J, K, L, M, or N. If it is one of these, the variable is typed INTEGER; otherwise it is REAL. If the variable appears in an executable statement, the "used" flag is set.

2.2.45 SUBROUTINE INIT. This subroutine processes assignment statements and statement functions for errors in syntax and transforms these statements in accordance with the rules outlined in Section 2.1.3. INIT recognizes the following four variations of the model statement "variable=expression".

Case A - the variable is unsubscripted
Case B - the variable is the function name within the function defining subprogram
Case C - the variable is subscripted
Case D - the statement is a function defining statement

1. Get assigned variable. If a left parenthesis follows, go to step 4.
2. If an equal sign follows, do one of the following:
   a. If variable is designated a function name, check that it is case B.
   b. If variable is not a function name, store variable in symbol table. (This is case A).
3. Set left-side type and go to Step 6.
4. If this is a function defining statement (case D),
   a. store the function name in the symbol table
   b. set type and change statement class
   c. store in statement function table
   d. set statement function flag
   e. store number of arguments.
5. If this is a dimensioned variable (case C),
   a. set type
   b. CALL PARSE to process the left-hand side
   c. CALL BLKSTR to store basic blocks for flow analysis and go to Step 7.
6. Store variable in basic block table.
7. CALL PARSE to process the right-hand side and store basic blocks.
8. CALL FNCSTR to process function references.
9. CALL EXPRCK to check validity of assignment.
10. CALL BLKSTR to store basic blocks and adjust order of variables.
11. If operating under the variable precision mode, CALL CNVRT, if necessary, to insert function references (Q1REAL, Q1COMP, Q1DPRE).
12. If operating under the roll call mode, and if the referenced function is a function module, generate call to ROLCHK.

2.2.46 SUBROUTINE INTRIN. After processing a program unit, this subroutine is called to check if any basic external or intrinsic function names have been misused.

1. Fetch the basic external or intrinsic function name from the Interface Definition file.
2. Search symbol table for name.
3. If found and the name is in variable or COMMON block list, issue diagnostic.

2.2.47 SUBROUTINE IO. This subroutine processes READ and WRITE statements.

1. Check keyword spelling.
2. Check that the I/O device is a simple integer variable.
3. Process formatted I/O statements.
4. If a FORMAT statement reference is made, fetch the statement number and flag it in the statement number table as having been referenced.
5. Process FORMAT arrays.
6. CALL EXPR and CALL PARSE to process the list.
7. CALL IOSTR to store basic blocks.
8. If operating under the roll call mode, transform READ statements into comment statements. If a READ statement is labeled, generate a CONTINUE statement with the same label.

2.2.48 SUBROUTINE IOSTR. This subroutine stores the basic blocks after an I/O list is encountered in a READ or WRITE statement. Variables which occur in I/O lists fall into three classes:

0 - I/O variable
1 - Subscript or implied DO parameter
2 - Implied DO index

1. Get class of variable.
2. If class 1, store variable as referenced.
3. If class 0, store variable as defined for a READ statement and as referenced for a WRITE statement.
4. If class 2, store as such and adjust its position in the basic block table.

2.2.49 FUNCTION IPREV. Given the character string A and a starting point IA in the string, this function returns the value 1, 2, or 3, depending upon whether the first preceding non-blank character is a digit, a letter, or some other character, respectively.

2.2.50 FUNCTION ITYPE. Given the character string A and a starting point ID, this function returns the value 1, 2, or 3, depending on whether the next non-blank character in A is a letter, a digit, or some other character, respectively.

2.2.51 SUBROUTINE LOGCHK. Given a character string in array A and a starting point LOGST in that array, this subroutine determines whether that which follows is a valid logical

constant or operator or relational operator. The variable LOG is set to 1 if the element is valid, otherwise it is set to zero. The variable LOGID is set to an integer between 1 and 11, depending on the operator or constant encountered, as shown in Figure 2-08.

OPERATOR/CONSTANT	LOGID
.LT.	1
.LE.	2
.GT.	3
.GE.	4
.EQ.	5
.NE.	6
.OR.	7
.AND.	8
.NOT.	9
.TRUE.	10
.FALSE.	11

FIGURE 2-08. Valid Operator/Constant Codes

2.2.52 SUBROUTINE LOGIF. This subroutine processes logical IF statements.

1. Check keyword spelling.
2. CALL PARSE to process the logical expression and store function references and basic blocks.
3. Check the legality of the statement associated with the logical IF.
4. Transfer control to the appropriate segment of code for each statement type.
5. Process assignment statements.
6. Process ASSIGN statements.
7. In the case of GO TO, assigned or computed GO TO, arithmetic IF, STOP, and RETURN statements, terminate the basic block and adjust the branch counter. In each case, control could pass to the next block, so a "998" flag is inserted (see FUNCTION NXTBLK).
8. Process GO TO statements. The number of branches out of the block is 2.
9. Process assigned GO TO statements.
10. Process computed GO TO statements.
11. Process arithmetic IF statements.
12. For Steps 9, 10, and 11, the number of branches out of the block is incremented by one.
13. Process CONTINUE, STOP, and RETURN statements. For STOP and RETURN, set the branch counter to 2.

2.2.53 SUBROUTINE LOOPCK. After each program unit has been processed, this subroutine is called to examine the basic block table to ensure that no transfer of control within the program unit violates the rule regarding branching into the range of a DO from outside its range. To accomplish this, the table ISTACK (COMMON block DOLOOP) is used to store information regarding each DO statement. ISTACK is a 4 X N array, where N is the number of DO statements in the program unit. The first word in each column contains the index in the statement number table of the terminal statement of the DO. The second contains the value 1 if a DO terminal is encountered and 0 if not. When processing is completed the second word must contain a 1. The third word contains the index in the ISTACK array of the loop in which the DO is nested, and the fourth word points to the induction variable in the symbol table. In the following example,

```
      DO 5 I=1,3
      DO 6 J=1,4
      IF(I.EQ.J) GO TO 10
    6 CONTINUE
   10 DO 7 K=1,5
      DO 9 L=1,4
      IF(K.EQ.L) GO TO 7
    9 CONTINUE
    7 CONTINUE
    5 CONTINUE
```

ISTACK contains the following information:

- Word 1 - Indices in the statement number table of the terminal statements for DO 5, 6, 7, and 9.
- Word 2 - All zeros before processing and all 1's after processing.
- Word 3 - The numbers 0, 1, 1, and 3.
- Word 4 - Indices in the symbol table for variables I, J, K, and L.

1. Test for entries in ISTACK.
2. Initialize IBLKST and IBLKND to the indices of the beginning and the end, respectively, of the first basic block.
3. Fetch the index in ISTACK of the DO which contains this basic block.
4. Fetch the number of branches from this basic block.
5. Test whether or not this is the last basic block.
6. Set the variable IST to the index in the basic block table of the first branch.
7. Search for the "998" flag (see FUNCTION NXTBLK) indicating a special type of branch which passes control to the next block.
8. Fetch the index of the basic block to which the branch under consideration branches.

9. Fetch from ISTACK the index of the DO loop which contains the previously identified basic block.
10. Test whether the index is zero, meaning that the basic block is not contained in a DO loop. In this case the transfer is all right.
11. Test whether the branch originated outside a DO loop and terminated inside a loop.
12. Test whether the branch began and ended in the same DO loop. If so, go to statement number 100 for testing the next branch from this basic block.
13. Move JLOOP up the DO stack until either the branch origin and the termination point are found to be the same DO loop (JLOOP=KLOOP) or the branch origin is found to be outside the range of a DO (JLOOP=0). The latter case signals improper DO nesting.

2.2.54 SUBROUTINE LVDLET. This subroutine deletes an entire function or the IPOSth value of the ITYPth type from the top or bottom (depending on the sign of IPOS) of a list of the requested function. IPOS and ITYP are explained in COMMON block LVARGS.

2.2.55 SUBROUTINE LVEXIT. This subroutine restores the syntax graph if GIRS memory becomes full.

2.2.56 SUBROUTINE LVFECH. This subroutine reads pertinent GIRS system variables plus the GIRS buffer containing the syntax graph structure.

2.2.57 SUBROUTINE LVFIND. This subroutine retrieves the IPOSth value of the ITYPth type from the top or bottom (depending on the sign of IPOS) of a list of values of a specified function. IPOS and ITYP are explained in COMMON block LVARGS.

LVFIND traverses multivalue lists sequentially so that N calls to LVFIND, to access the first through the nth items on a multivalue list, result in N(N+1)/2 accesses to core. A saved index facility reduces this number of accesses to N at the cost of four words of core for every call to LVFIND. These words take the form of four separate variables in the calling sequence for each call to LVFIND with a different node-link pair (IFUNC, IARG) as input. The LVFIND arguments must be initialized to zero before the first call to LVFIND for the associated IFUNC and IARG and must not be changed thereafter by the user program. If the saved index option is not being used, the function, argument, type of data desired, and other input required in the COMMON block LVARGS is initialized and LVFIND is executed with a dummy variable as its four arguments. If the saved index option is being used, the arguments are a set of four completely different variables for each separate function-argument pair (IFUNC, IARG) that are input.

2.2.58 SUBROUTINE LVGRN. This subroutine assigns a unique random number to a given GIRS identifier. Each time LVGRN is referenced, it generates a different integer between one and MEMSZE (the length of the GIRS buffer). An attempt to define more than MEMSZE identifiers will terminate the program.

2.2.59 SUBROUTINE LVNSRT. This subroutine places a triple into a GIRL structure.

2.2.60 SUBROUTINE LVSETP. This subroutine initializes the four fields in the GIRS buffer, the variables needed for SUBROUTINE LVGRN, and the register of available space.

2.2.61 SUBROUTINE MODID. This subroutine is used when operating in the roll call mode to check the program unit's roll call actions. MODID is referenced by the main roll call mode subroutine GENROL. It prints out what is written on each output device (X, Y, and Z). MODID is only referenced for a roll call mode equal to 2.

2.2.62 FUNCTION NEXT. Given the character string A of length N and a starting point IA, this subroutine sets the value of the function to the next non-blank character found. JPTR is advanced to the next position in the array. If no more non-blank characters are found, NEXT returns with the value 'blank' and sets the string pointer JPTR to N+1.

2.2.63 FUNCTION NXTBLK. Given a location in the basic block table, which is a branch to a block, this function returns the location of the block which the branch points to.

Case 1 - Branch is a "999", indicating RETURN or STOP. Set NXTBLK=0 and return.
Case 2 - Branch is a "998", indicating a branch to the next basic block. Compute starting location of next block.
Case 3 - Branch is to a statement label. Fetch starting location of block from the statement number table.

2.2.64 SUBROUTINE PARSE. This subroutine processes assignment, CALL, and I/O statements as a series of terminal symbols. If it is possible to trace through the syntax graph from a start to a stop state, then the statement is declared to be syntactically correct.

1. CALL subroutines PHONEY and LVFECH to read the syntax graph and initialize GIRL variables.
2. Initialize appropriate variables for parsing the next statement.
3. Zero out the parser tables.

4. Store the expression to be parsed (created by SUBROUTINE EXPR) in a list structure so that it will be in a form suitable for parsing.
5. CALL RECOG to perform the parse.
6. CALL PRNTS to generate error messages.
7. Delete all GIRL structures which were created by the parse.

2.2.65 SUBROUTINE PHONEY. This subroutine initializes GIRL variables.

2.2.66 SUBROUTINE PRNTS. To guard against the tokens (nodes and links) already recognized being lost when a syntax error occurs, this subroutine places each token recognized so far into a buffer for printing.

2.2.67 SUBROUTINE PROG. This subroutine is only used when a main program is being analyzed. For the CDC version of AUDIT, PROG processes the PROGRAM statement, which must precede a main program on the CDC. The syntax of the PROGRAM statement is checked and the main program name is stored in the symbol table. For the IBM 360 and UNIVAC 1108 versions of AUDIT, PROG assigns the name MAIN to a main program and enters that name in the symbol table.

2.2.68 SUBROUTINES Q1COMP, Q1DPRE, and Q1REAL. These packages of subroutines, available as card input, are used when operating under the variable precision mode. Q1COMP, Q1DPRE, and Q1REAL perform truncation during operations between complex, double precision, and real numbers, respectively. Each package contains one version of each of the three subprograms which will truncate the n rightmost bits during each operation. For each value of n for which the user wishes to perform the variable precision calculation, a different version of a package must be used. Each package consists of the three subprograms for a single bit configuration. There is a separate and different set of packages for each AUDIT processor. For the CDC 6000, there are 11 packages that simulate word lengths ranging from 30 to 40 bits. For the UNIVAC 1108, there are six packages that simulate word lengths ranging from 30 to 35 bits. The IBM 360 has eight packages that simulate word lengths ranging from 24 to 31 bits.

The CDC has a 60-bit word consisting of a 12-bit exponent and a 48-bit fraction. Each package of variable precision functions simulates a certain word length by masking the appropriate number of low order (rightmost) bits from the fraction part of the word. For the CDC, a 30-bit word length is simulated by masking 30 bits. A 40-bit word length is simulated by masking 20 bits. For the UNIVAC 1108 and the IBM 360, the fraction part of the word is masked in the same manner.

2.2.69 SUBROUTINE REALCK. Given the character string A and a starting point IP, this subroutine determines whether or not the next substring is a real number. REALCK sets IREAL to 1 if the substring is a real number and 0 if it is not.

2.2.70 SUBROUTINE RECOG. This subroutine checks the syntax of the input string against the syntax graph.

2.2.71 SUBROUTINE RECOV. This subroutine provides for backup recovery. FORTRAN is a non-finite state language, meaning that there may be more than one link eminating from a source node. If the wrong path has been selected while tracing through the syntax graph, RECOV is referenced to back up the trace to the correct node, so that a different path can be selected.

2.2.72 PROGRAM ROLCAL. For the CDC version of AUDIT, ROLCAL is the name assigned to the main program generated by subroutine GENROL (see Section 2.2.37). For the IBM 360 and UNIVAC 1108 versions of AUDIT, this main program is generated without a name and without a PROGRAM statement. See subroutine GENROL for an explanation of this main program.

2.2.73 SUBROUTINE ROLCHK. This subroutine packs the characters from Hollerith variables of the argument list into one word, and writes it onto logical unit 3. Each of the six argument list variables contain one character. The six characters (some may be blank) make up a subprogram name. The characters in the argument list are generated by SUBROUTINE CALL2.

2.2.74 SUBROUTINE SEARCH. Given the identifier NXTID, this subroutine searches the symbol table for an occurrence of the identifier and sets ISRCH(1) to 1 if the identifier occurs as a variable name and 0 otherwise. ISRCH(2) is set to 1 if the identifier occurs as a subprogram and 0 otherwise.

2.2.75 SUBROUTINE SEMANT. This subroutine checks a statement for correct semantics. For example, if the variable A is dimensioned A(10), the statement A(B)=1. is syntactically correct but semantically incorrect, since B is real instead of integer. The parser will accept any constant or variable as a subscript, therefore an error in the type of dimensionality of a subscript must be checked by a semantics routine.

1. Check for mixed mode expressions.
2. Check for correctness of subscripts.
3. Insert call to variable precision functions.
4. Keep track of function call levels.
5. Build the parser table.

2.2.76 SUBROUTINE SEPAR. Given the starting point SEPST in the string A, this subroutine sets the variable ISEP to 1, 0, or -1, depending on whether the next substring is a valid separator, is not a separator, or is an invalid separator, respectively. A valid separator is a comma, a slash, or a series of slashes.

2.2.77 SUBROUTINE SIMP. This subroutine checks the RETURN, STOP, CONTINUE, and BLOCK DATA statements for correct spelling. For the RETURN and STOP statements, the basic block (associated with flow analysis) in which they occur is terminated.

2.2.78 SUBROUTINE SLEVEL. While tracing through the syntax graph, this subroutine stacks the nodes that were seen, so that a back up action can be performed if necessary.

2.2.79 SUBROUTINE SQUEEZ. Given a character string of length M in array D, this subroutine removes all blanks and adjusts M accordingly. Hollerith fields are not affected.

2.2.80 SUBROUTINE SSTOP. After a string has been completely traced, this subroutine verifies that the current state is a valid final state.

2.2.81 SUBROUTINE STATNO. This subroutine performs the following functions:

a. Checks to see that the statement being processed is labeled and that the label is valid.
b. Determines whether the statement should begin a new basic block. If so, STATNO closes the old block and initiates a new one.
c. Checks for proper DO loop nesting.

1. Check for the presence of a statement label.
2. Process unlabeled statements.
3. Process END statements; the END statement must be preceded by some type of branch statement (except a logical IF) or a RETURN or STOP statement (BLOCK DATA subprograms are excepted).
4. Store the number of branches contained in this program unit in the last basic block. This concludes the processing for END statements.
5. Detect FORMAT statements.
6. If the statement being processed is the first executable statement, go to step 9. If the preceding statement ended a basic block, go to step 17.
7. If the preceding statement was the terminal statement of a DO loop, go to step 16.
8. If the preceding statement contained transfers to other blocks but not this one, issue a diagnostic.

9. Begin processing the first executable statement. Initialize the basic block table and go to Step 13.
10. Process the labeled statement. If the labeled statement is an END statement, issue a diagnostic.
11. Check the statement number to see that it occupies the proper position in the statement.
12. Check for duplicate statement numbers. Set the "defined" flag in the statement number table and store the statement type in the statement number table.
13. If the statement being processed is a FORMAT statement, RETURN.
14. If this labeled statement is the first executable statement in the program unit, go to Step 9.
15. If the previous statement ended a basic block, go to Step 17.
16. Increment the statement count for this block. Store a "998" flag in the basic block table to signify that control may pass to this new block from the previous block. Set the branch counter into the block to 1. See FUNCTION NXTBLK.
17. Close out old basic blocks and initialize the new one.
18. Set the start of the new block.
19. Store the pointer to the new block and the branch count in the old block.
20. If the preceding statement is the terminal statement of a DO loop, make some adjustments so that the induction variable becomes undefined at the end of that statement rather than at the beginning.
21. Reinitialize the branch counter NBRNCH.
22. Store the number of the current DO loop in the new basic block.
23. If the statement is not labeled, RETURN.
24. Store the index of the basic block headed by a statement label in the statement number table entry corresponding to that label.
25. Process statements which terminate DO loops.
26. If this statement does not end the current loop, issue a diagnostic.
27. Check that the proper type of statement terminates the DO loop.
28. Set an entry in the DO stack to indicate that the current DO loop is complete.
29. Flag the induction variable in the DO loop as undefined.
30. Search the DO stack for the first unsatisfied DO, which then becomes the current loop.
31. If there are no further unsatisfied DO loops, set ILOOP to zero and RETURN.
32. Check whether more than one DO is terminated by this statement.
33. Make appropriate changes in the DO stack and set the induction variable as undefined.

2.2.82 SUBROUTINE STFNC. When a statement function reference is encountered, this subroutine checks the calling sequence against the argument list in the function defining statement.

1. Fetch the number of arguments from the symbol table.
2. Fetch the number of arguments encountered by the parser and verify the correctness.
3. Check arguments for proper dimensionality (must be zero) and proper type.

2.2.83 SUBROUTINE STORE. Given the identifier NXTID and a class (IDTYP=1 for variable, IDTYP=2 for subprogram, IDTYP=3 for COMMON block, this subroutine stores the identifier in the next available location in the symbol table and links it to the last identifier of that class.

2.2.84 SUBROUTINE STSRCH. Given a statement number N2, this subroutine searches the statement number table STATRA for that statement number, and stores it in the table if it is not already there. In any case, the index of the label is stored in LOC.

2.2.85 SUBROUTINE SUB. This subroutine processes SUBROUTINE and FUNCTION statements.

1. Check keyword spelling.
2. Process the list of formal parameters by storing the names in the symbol table.
3. Determine that there are no more than 63 arguments.
4. If function is typed, store type in the symbol table.

2.2.86 SUBROUTINE SUBCHK. After the program unit has been processed, this subroutine checks its name and argument list against the Interface Definition file.

1. Increment subroutine counter and check for overflow.
2. Fetch the number of arguments and their types from the symbol table.
3. If the subprogram name has already been encountered, go to step 7.
4. Search the Interface Definition file for the subprogram name.
5. If found, store file location in symbol table and go to step 8.
6. If not found, issue a diagnostic and store the name in the temporary Interface Definition file. Create and store an Interface Definition for the subprogram based on this occurrence (stored in temporary file).
7. Fetch the Interface Definition file location of subprogram.

8. Fetch the number of arguments and program unit type and class, and check validity.
9. Check the variables in the argument list for correct type, dimensionality, and I/O status.

2.2.87 SUBROUTINE SWITCH. This subroutine takes an identifier out of the variable name list and puts it into the function name list in the symbol table. The need for this action arises when a statement of the type A=F(13) follows a statement of the type INTEGER F. F is initially put on the variable list and the second statement requires that it be put on the function list.

1. Find the variable entry which points to F.
2. Link this variable to the variable following F.
3. Link F into the function list.

2.2.88 SUBROUTINE SYMTAB. This subroutine displays the symbol table.

1. Print headings.
2. Fetch the next variable and its type, dimensionality, and relocation from the symbol table and display it.
3. After all variables have been displayed, fetch the next external from the symbol table along with its type and number of arguments and display it.
4. After all externals have been displayed, fetch the next statement function from the statement function list and fetch its type and the number of arguments from the symbol table and display it.
5. After all statement functions have been displayed, display all statement labels.
6. After displaying all statement labels, fetch COMMON blocks and their lengths from the symbol table and display them.

2.2.89 SUBROUTINE TYPE. This subroutine processes the INTEGER, REAL, DOUBLE PRECISION, COMPLEX, and LOGICAL statements by first checking keyword spelling and then adding to the symbol table the type and dimension information contained in the list. Considerable error checking is performed including the detection of array bounds that are outside of allowable range, the illegal use of variable dimension in TYPE statements, dimensioning previously dimensioned variables, typing previously typed variables, and other illegal uses of dimensioned variables.

2.2.90 Blank COMMON Storage.

1. A(1326) - This array holds, in A1 format, the FORTRAN statement currently being processed. For most of the program unit descriptions, this array is referred to as the character string A.

2. D(500) - This array is used to store the result of a call to subroutine GNLE. After such a call, array D contains, in A1 format, the next operator, identifier, constant, or separator in the FORTRAN statement being processed.

3. IDTBL(8,500) or IDTBL(11,500) - This symbol table array contains information regarding the symbols used in the program unit being processed. These symbols fall into the following three classes: COMMON block names, variable names, and subprogram names. IDTBL (8,500) is used for the CDC and UNIVAC 1108 versions of AUDIT. IDTBL (11,500) is used for the IBM 360 version of AUDIT. Up to 500 symbols may be entered into the IDTBL array. For each symbol, the eight or eleven columns of the array are used. IDTBL(8,500) for the CDC is constructed as follows:

    Word 1 - Contains the symbolic name in A6 format.

    Word 2 - Contains a pointer to the next symbol in the class of which this is a member.

    Word 3 - Contains flags and information about the symbol:

    Bit 1 is 1 if the symbol is dimensioned and 0 if not.

    Bits 2-7 contain the number of dimensions or the number of arguments depending on whether the symbol is a variable or a subprogram.

    Bits 8-10 contain the variable or function type (1 for real, 2 for complex, 3 for double precision, 4 for integer, 5 for logical).

    Bit 11 is 1 if the symbol has been typed and 0 if not.

    Bit 12 is 1 if the symbol is a formal parameter and 0 if not.

    Bit 13 is 1 if the symbol is a variable dimension and 0 if not.

    Bit 14 is 1 if the symbol has been declared in a DATA statement and 0 if not.

    Bit 15 is used by subroutine FLOWCK to suppress printing of multiple error messages.

    Bit 16 is 1 if the symbol is in COMMON and 0 if not.

    Bit 17 is 1 if the symbol is equivalenced and 0 if not.

    Bit 18 is 1 if the symbol's Interface Definition file location is known and 0 if not. (in this case the symbol must be a subprogram name).

    Bit 19 is 1 if the symbol is a statement function name and 0 if not.

Bits 19-36 contain the first subscript (if the symbol is an array name) or contain the symbol's Interface Definition file location (if the symbol is a subprogram name).

Bit 37 is 1 if the symbol (must be a formal parameter) is input and 0 if not.

Bit 38 is 1 if the symbol has appeared in an executable statement and 0 if not.

Word 4 - If the symbol is a variable which is an array, this word contains the second and third dimensions of the array (if any) in two consecutive 18-bit fields, respectively. If the symbol is a COMMON block name, this word contains the size of the block.

Words 5 and 6 - Contain COMMON block information. If the symbol is a COMMON block name, Words 5 and 6 point to the first and last names in the block, respectively. If the symbol is the name of a variable in COMMON, Word 5 points to the next variable in the block and Word 6 points to the name of the block.

Words 7 and 8 - Contain EQUIVALENCE information. If the symbol has been declared in an EQUIVALENCE statement, Word 7 points to the next variable in the chain and Word 8 contains the offset.

IDTBL(8,500) for the UNIVAC 1108 is constructed in the same manner as for the CDC except for the following exceptions.

Word 3:

1. Bit 15 is the same as bit 37 of the CDC.
2. Bit 19 is the same as bit 38 of the CDC.
3. Bit 20 is the same as bit 19 of the CDC.
4. Bits 20-36 are the same as bits 19-36 of the CDC. Note that bit 20 is used twice but will never conflict with the other use.

Word 6 is also used in the same manner as bit 15 of the CDC for word 3.

IDTBL(11,500) for the IBM 360 is constructed as follows:

Words 1 and 2 - Contain the symbolic name in A4, A2 format.

Word 3 - Contains flags and information about the symbol.

Bits 1-14 . See bits 1-14 of IDTBL(8,500) for the CDC.

Bit 15 is 1 if the symbol (must be a formal parameter) is input and 0 if not.

Bits 16-19. See bits 16-19 of IDTBL(8,500) for the CDC.
Bit 20 is used by subroutine FLOWCK to suppress printing of multiple error messages.
Bit 21 is 1 if the symbol has appeared in an executable statement and 0 if not.
Bits 22-23 are not used.
Bits 24-32 contain the symbol's Interface Definition file location (if the symbol is a subprogram name).

Word 4 - See Word 2 of IDTBL(8,500) for the CDC.

Words 5-7 contain the first, second, and third subscripts (if the symbol is a variable which is an array).

Words 8-9 - See Words 5 and 6 of IDTBL(8,500) for the CDC.

Words 10-11 - See Words 7 and 8 of IDTBL(8,500) for the CDC.

4. INITID(3) - This array contains pointers to the IDTBL array (symbol table) for the first entries in each of the symbol classes: variable, subprogram, and COMMON block name, respectively.

5. LASTID(3) - This array contains pointers to the IDTBL array (symbol table) for the last entries in each of the symbol classes: variable, subprogram, and COMMON block name, respectively.

6. ISRCH(3) - Each member of this array contains 1 or 0, depending on whether or not a symbol is the name of a variable, subprogram, or a COMMON block.

7. JPTR - This integer variable points to the current character position in array A.

8. N - The number of characters in the statement being processed.

9. M - The length in characters of the language element (identifier, operator, separator, constant) just identified by subroutine GNLE.

10. JTYP - Set by subroutine GNLE according to the kind of language element encountered in the input string. See Figure 2-07.

11. LSTART - Points to the beginning of the language element in array A just recognized by subroutine GNLE.

12. N2 - The binary equivalent of the character string just identified by subroutine GNLE as an integer.

13. IFNCNM - Contains the function name in A6 format (for the CDC and UNIVAC 1108 versions) or the symbol table location of the function name (for the IBM 360), if the program unit being analyzed is a function subprogram.

14. LOGID - Set by subroutine LOGCHK; denotes the integer code for the logical operator of the character string currently being considered. See Figure 2-08.

15. NXTID or NXTID(2) - The identifier currently being considered; A6 format in NXTID (used for CDC and UNIVAC 1108), and A4 and A2 format in NXTID(2) (used for IBM 360).

16. IDTYP - An indicator of the class (1 for variable, 2 for subprogram, 3 for COMMON block) to which the identifier in NXTID will be attached. IDTYP is used as input to subroutine STORE.

17. NID - A count of the number of entries in the symbol table.

18. LOC - Normally contains a pointer into the symbol table to the identifier which has just been stored by subroutine STORE.

19. LTYP - Set to 9 if the statement being processed is a logical IF statement. The statement following the logical IF is classified by variable ITYP.

20. ITYP - Set by subroutine CLASS to indicate what kind of FORTRAN statement is in array A. See Figure 2-04.

21. IBLKDT - Set to 1 by the main program if the program unit being analyzed is a BLOCK DATA subprogram and 0 if not.

22. MODE - This value is submitted as input; it indicates which mode is to be used:
    1 - Variable precision mode and/or audit mode
    2,3 - Roll call mode.

23. IERR - An error flag set by subroutine BUILD:
    1 - Too many continuation statements
    2 - An end-of-file has been encountered.

24. IDES - Subroutine REALCK sets IDES to 1 if a real number is double precision and 0 if it is not. Subroutine DESCRP sets IDES to 1 if a format field descriptor is valid and 0 if it is not.

2.2.91 COMMON Block BASBLK.

1. IBLOCK(2500) - This array (the basic block table) contains information regarding the definition of variables and the flow of control through the program unit.
2. NBLOCK - Counter which keeps track of the number of entries in the basic block table.
3. NB - Set to 1 when a new basic block is started.
4. NBRNCH - On a block terminal statement, set to the number of branches from the block.

During the processing of each program unit, the basic blocks which represent the program unit structure are constructed. A basic block contains the information represented by a group of statements which physically appear together in the program, are located within the same DO loop, and are executed together. See Section 2.1.4 for an explanation of what begins and what terminates a basic block. The basic block table (IBLOCK(2500)) usually contains information on more than one block (except for a very simple program unit). The basic blocks are of variable length and each block points to the starting location of the next block.

The first word (location) for each block is the "head of the block". It contains the following information:

Bits		Contents
IBM 360	CDC, UNIVAC 1108	
1-6	1-6	A count of the number of possible branches from the block.
7-12	7-12	A pointer into ISTACK (see COMMON block DOLOOP) to the DO loop containing the block. This field is zero if the block is not within a DO loop.
13-24	13-28	A pointer to the next block.
25-32	29-36	A pointer into STATRA (see COMMON block LABELS) to the statement number of the block. This field is zero if the block has no statement number.

The locations following the "head of the block" contain information about the variables in the block, stored one variable per word according to the following scheme:

1001≤n≤1500: n-1000 points to the symbol table location (see IDTBL array in blank COMMON) of the variable. The variable is defined by this block.
2001≤n≤2500: n-2000 points to the symbol table location of the variable. The variable is referenced by this block.
3001≤n≤3500: n-3000 points to the symbol table location of the variable. The variable is an induction variable in this block.
4001≤n≤4500: n-4000 points to the symbol table location of the variable. The variable is assigned a value (by an ASSIGN statement) in this block.
5001≤n≤5500: n-5000 points to the symbol table location of the variable. The variable appears in an assigned GO TO in this block.
6001≤n≤6500: n-6000 points to the symbol table location of the variable. The variable is undefined in this block.
7001≤n≤7500: n-7000 points to the symbol table location of the variable. The variable is a DO parameter.

The remaining locations of the block contain information about branches from the block, stored one variable per word according to the following scheme:

1≤n≤200: branch to the basic block pointed to by n, which is a location in the statement number table (see COMMON block LABELS). Once in the statement number table find the block pointed to.
n=998: branch to next block
n=999: STOP or RETURN

The IBLOCK array is structured as follows:

Word 1: Head of block 1
Words 2-M: Variable info. for block 1
Words (M+1)-N: Branch info. for block 1
Word N+1: Head of block 2
Words (N+1)-L: Variable info. for block 2
Words (L+1)-J: Branch info. for block 2
Word J+1: Head of block 3
etc.

Words 1, N+1, and J+1 are each separated into four fields (see Word 1 description) which give the head of block information. All other words contain the variable n as described in the preceding paragraphs.

To illustrate the construction of the IBLOCK array, the following program unit is analyzed. Items needed for the subroutine - symbol table (IDTBL array in blank COMMON), statement number table (STATRA array in COMMON block LABELS), DO loop stack (ISTACK array in COMMON block DOLOOP), basic blocks, and the basic block table (IBLOCK array in COMMON block BASBLK) - are included.

SAMPLE PROGRAM UNIT

BASIC BLOCK NUMBER			STATEMENT
		SUBROUTINE SORTS(NUM,N)	1
		DIMENSION NUM(1),IPS(10)	2
		DATA (IPS(I),I=1,10)/10,9,8,7,6,5,4,3, 2,1/	3
1		DO 1 II=1,10	4
2		IF(N.GT.IPS(II)) GO TO 2	5
3	1	CONTINUE	6
4	2	JJ=II+1	7
		DO 10 NN=JJ,10	8
5		IP=IPS(NN)	9
		DO 20 I=1,IP	10
6		K=N/IP	11
		IP(I+K*IP-N) 21,21,22	12
7	22	K=K-1	13
8	21	DO 20 J=1,K	14
9		M=I+IP*(J-1)	15
		MM=M+IP	16
10	30	IF(NUM(M)-NUM(MM)) 20,20,23	17
11	23	ITEMP=NUM(M)	18
		NUM(M)=NUM(MM)	19
		NUM(MM)=ITEMP	20
		IF(M-I) 20,20,24	21
12	24	MM=M	22
		M=M-IP	23
		GO TO 30	24
13	20	CONTINUE	25
14	10	CONTINUE	26
		RETURN	27
		END	28

SYMBOL TABLE

SECOND SUBSCRIPT	NAME
1	SORTS
2	NUM
3	N
4	IPS
5	I
6	II
7	JJ
8	NN
9	IP
10	K
11	J
12	M
13	MM
14	ITEMP

STATEMENT NUMBER TABLE

SECOND SUBSCRIPT	STATEMENT NUMBER
1	1
2	2
3	10
4	20
5	21
6	22
7	30
8	23
9	24

DO LOOP STACK

SECOND SUBSCRIPT	POINTER TO STATEMENT NUMBER TABLE	POINTER TO NEST	POINTER TO SYMBOL TABLE
1	1	0	6
2	3	0	8
3	4	2	5
4	4	3	11

There are 14 basic blocks. The following table indicates which statements belong to which basic block.

BASIC BLOCK	STATEMENT
1	4
2	5
3	6
4	7,8
5	9,10
6	11,12
7	13
8	14
9	15,16
10	17
11	18,19,20,21
12	22,23,24
13	25
14	26,27,28

BASIC BLOCK TABLE

LOCATION	CONTENTS	LOCATION	CONTENTS
1	1 0 4 0	49	2009
2	3006	50	1013
3	998	51	998
4	2 1 10 0	52	2 4 59 7
5	2003	53	2002
6	2004	54	2012
7	2006	55	2002
8	2	56	2013
9	998	57	4
10	1 1 12 1	58	8
11	998	59	2 4 74 8
12	1 0 18 2	60	2002
13	2006	61	2012
14	1007	62	1014
15	3008	63	2002
16	7007	64	2013
17	998	65	2012
18	1 2 25 0	66	1002
19	2004	67	2014
20	2008	68	2013
21	1009	69	1002
22	3005	70	2012
23	7009	71	2009
24	998	72	4
25	2 3 35 0	73	9
26	2003	74	1 4 81 9
27	2009	75	2012
28	1010	76	1013
29	2005	77	2012
30	2010	78	2009
31	2009	79	1012
32	2003	80	7
33	5	81	1 4 83 4
34	6	82	998
35	1 3 39 6	83	1 2 0 3
36	2010	84	99
37	1010		
38	998		
39	1 3 43 5		
40	3011		
41	7010		
42	998		
43	1 4 51 0		
44	2005		
45	2009		
46	2011		
47	1012		
48	2012		

Sample program unit SUBROUTINE SORTS has 28 statements and 14 basic blocks. There are 14 symbolic names used in SORTS. These 14 names are stored in the symbol table. Assuming IDTBL(8,500) is used, all information on the symbolic name IPS, for example, is stored in IDTBL(1,4), IDTBL(2,4),.....,IDTBL(8,4). IPS is the fourth name stored in the IDTBL array. There are nine statement numbers stored in the statement number table. All information on statement number 30, for example, is stored in STATRA(1,7) and STATRA (2,7), since statement number 30 is the seventh statement number found. There are four DO loops stored in the DO loop stack. All information on DO 10, for example, is stored in ISTACK(1,2), ISTACK(2,2), ISTACK(3,2), and ISTACK(4,2), since DO 10 is the second DO loop encountered. ISTACK(1,2) contains the integer 3, which points to the third statement number in the statement number table. The third statement number is 10, which is the terminal statement number of the DO. ISTACK(3,2) contains the integer 0, which means this DO is not contained (nested) in any other DO. ISTACK(4,2) contains the integer 8, which points to the eighth name in the symbol table. The eighth name is NN which is the induction variable for this DO.

The 14 basic blocks comprise statements 4 through 28. Basic block 11, for example, consists of statements 18,19,20, and 21. The basic block table, IBLOCK(2500), contains all information on the basic blocks. IBLOCK of 1-3 describes basic block 1, IBLOCK of 4-9 describes basic block 2, etc. Basic block 14, for example, is described by IBLOCK of 52-58. IBLOCK(52) contains four fields. The value in the first field, 2, indicates there are two branches from this block. The value in the second field, 4, points to the fourth DO loop in the DO loop stack indicating that this block is contained in the nested DO 20. The value in the third field, 59, indicates the location in IBLOCK of the next block. The value in the fourth field, 7, points to the seventh statement number in the statement number field indicating that this block has a statement number of 30. IBLOCK(53) and IBLOCK(55), each containing the number 2002, indicate that the variable is referenced ($2001 \leq 2002 \leq 2500$). 2002-2000 gives the number 2 which points to the second name in the symbol table, NUM. IBLOCK(53) and IBLOCK(55) both having the integer 2002 indicate that the variable NUM is twice referenced in this block. IBLOCK(54) indicates that the variable M (2002-2000=2, which points to M in the symbol table) is referenced ($2000 \leq 2002 \leq 2500$) in this block. IBLOCK(55) indicates that the variable MM is referenced in this block. IBLOCK(57) and IBLOCK(58), being equal to 4 and 8, respectively, point to the fourth and eight locations in the statement number table indicating that this block branches to statement numbers 20 and 23.

2.2.92 COMMON Block DOLOOP. This block contains information about the program DO loop structure.

1. ISTACK(4,50) - This integer array is constructed by subroutines DO and STATNO, which process DO statements and statement labels, respectively. The array stores the DO structure of the program unit for later checking. As many as 50 DO statements may be stored. There are four words that describe the DO structure for each DO loop:
   Word 1 - Contains a pointer to the statement label table entry (see COMMON block LABELS) for the terminal statement label.
   Word 2 - Used for later processing.
   Word 3 - Contains a pointer into ISTACK to the DO which contains the DO represented by this DO loop.
   Word 4 - Contains a pointer to the symbol table (see blank COMMON storage - IDTBL) for the induction variable.

2. NSTACK - The number of DO loops in the ISTACK array.

3. ILOOP - Identifies, for the statement being processed, which DO loop the statement is contained in.

4. IOVFLW - Set to 1 if the DO stack overflows; if it has, DO loop processing is terminated.

2.2.93 COMMON Block FLOW.

1. IFL - Flow analysis mode as input by the user:
   0 - no flow analysis
   1 - moderate flow analysis
   2 - full flow analysis
   IFL is set to -1 if the flow analysis cannot be performed due to error(s) in the program unit.

2. IRP - Repeat parameter (set to IFL-1) that indicates the number of times a statement number may be repeated in a given flow path.

2.2.94 COMMON Block FORMAT.

1. IDESST - Location within the input string where the scan for a format field descriptor is to start; set by subroutine GROUP.

2. IDESND - If subroutine DESCRP finds a valid field descriptor, this is the last location which it occupies in the string.

3. IGPST - Location within the input string where scan for a group of format field descriptors is to begin; set by subroutine FRMAT.

4. IGPND - If subroutine GROUP finds a valid group of field descriptors, this is the last location which it occupies in the string.

5. IGRP - Set to 1 if a group of field descriptors is valid and 0 otherwise.

6. SEPST - Location within the input string where scan for a format field descriptor is to begin; set by subroutine GROUP.

7. SEPND - If subroutine SEPAR finds a valid field separator, this is the last location which it occupies within the string.

8. DIR - Indicates, for subroutine SEPAR, which direction to search for a field separator (=1 for search forward, =-1 for search backward); set by subroutine GROUP.

9. ICOM - Set to 1 if a field separator is a comma and 0 if it is not.

10. ISEP - Set to 1, 0, or -1 for a valid field separator, not a field separator, or an invalid field separator, respectively.

2.2.95 COMMON Block FUNC.

1. IFNCRA(5,12) - This array contains information about function and subroutine references that occur in the statement. Each row of the array represents a different reference. The rows are arranged according to the order in which the references physically occur in the statement. IFNCRA is constructed by the PARSE subroutine.
   Word 1 - Contains the number of arguments encountered.
   Words 2-12 - Contain the type and dimensionality of the various arguments. Each word is stored in the same manner as it is stored in array INTFAC of COMMON block LIST. However, for the CDC version, Bits 55-60 of each word contain a value 1 if the arguments in fields 1-6, respectively, of this word are in an expression, and 0 otherwise. For the UNIVAC 1108 and IBM 360, this information is stored in the INTFAC array. The I/O status is not

determined in the IFNCRA array and is therefore set to zero for each argument.

2. MARGS - A running count of the number of entries in IARGS(50).

3. IARGS(50) - This array keeps track of all variables which occur in an expression and notes the relationship of the variables to any function references or to any implied DO loops within the statement. IARGS is constructed by the parser subroutines. Information is stored in 20-bit fields for the CDC version, in 18-bit fields for the UNIVAC 1108, and in fields of 18 bits (for arithmetic and logical expressions) and 16 bits (for I/O lists) for the IBM 360. A field is constructed for an arithmetic or logical expression, or for an I/O list.

For an arithmetic or logical expression, information is stored in 20-bit fields for the CDC version and in 18-bit fields for the UNIVAC 1108 and the IBM 360. The CDC has three fields per word, the UNIVAC 1108 two fields per word, and the IBM 360 one field per word. For arithmetic and logical expressions, a field is constructed as follows:

UNIVAC 1108 IBM 360 Bits	CDC Bits	
1-9	1-10	Symbol table location of variable.
10-12	11-13	The index of the function or subroutine of which this variable is an argument. This index points to the function or subroutine in IFNCRA(5, 12). The index is set to zero if the variable is not an argument of a function or a subroutine.
13-18	14-19	The number of the argument of the function or subroutine which this variable is associated with. Set to zero if variable is not an argument.
	20	Left blank.

For an I/O list information is stored in 20-bit fields for the CDC version, 18-bit fields for the UNIVAC 1108, and 16-bit fields for the IBM 360. The CDC version has three fields per word and the UNIVAC 1108 and IBM 360 have two fields per word. For I/O lists a field is constructed as follows:

UNIVAC 1108 Bits	IBM 360 Bits	CDC Bits	
1-9	1-9	1-10	Symbol table location of variable.
10-12	10-11	11-15	Set to 0, 1, or 2 depending on whether the variable is an I/O variable, subscript or implied DO parameter, or implied DO index, respectively.
13-18	12-16	15-20	If this variable is an implied DO index, this field contains the symbol table location of the first variable within the expression which is within the implied DO. Otherwise this field is set to zero.

4. FNCLOC(5) - This array keeps track of the symbol table location of subroutines or functions referenced within a statement.

5. NFUNC - Number of references to functions or subroutines within a statement.

To illustrate the construction of COMMON block FUNC, two examples are offered. A symbol table and arrays IFNCRA, FNCLOC, and IARGS for each example are constructed. Assume that the CDC version of AUDIT is being used.

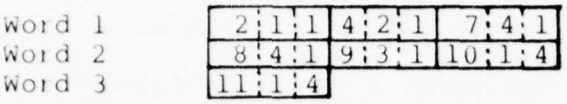
EXAMPLE 1

```
DIMENSION I(10,10)
CALL SUB(I,SIN(B),COS(ABS(C+D)-E),J+K)
```

Given the CALL SUB statement as above, the following arrays are constructed:

SYMBOL TABLE		IFNCRA			FNCLOC
			WORD 1	WORD 2	
1	SUB				1
2	I				3
3	SIN	Row 1	4	4:2:0 1:0:0 1:0:0 4:0:0	5
4	B	Row 2	1	1:0:0	6
5	COS	Row 3	1	1:0:0	
6	ABS	Row 4	1	1:0:0	
7	C				
8	D				
9	E				
10	J				
11	K				

IARGS (for CDC)



Information on the CALL SUB statement is stored in the symbol table, and the IFNCRA, FNCLOC, and IARGS arrays. Symbolic names are stored in the symbol table in the order of their occurrence in the CALL statement. (Even though I should be first in the symbol table, since it appears in the DIMENSION statement, for simplicity just be concerned with the order of names in the CALL statement.) Seven variables are used as arguments within the statement. I, J, and K are arguments of SUB, B is an argument of SIN, C and D are arguments of ABS, and E is an argument of COS. There is one subroutine reference, one intrinsic function reference, and two basic eternal function references. Four rows of IFNCRA are therefore used to describe seven variables. Rows 2, 3, and 4 each describe the single arguments of SIN, COS, and ABS, respectively. Row 1 describes the four arguments which are variables for the SUB reference. For Row 1, Word 1 contains the number of arguments (4). Word 2 consists of four fields (for the four arguments), each field containing three numbers. The first field of Word 2, for example, indicates that the first variable is of integer type (4) and is double subscripted(2). The FNCLOC array contains pointers, to the symbol table, of each subroutine and function name referenced in the statement. FNCLOC(4)=6 indicates that the fourth reference in the statement has the symbolic name ABS. The IARGS array contains information on each of the seven variables in the statement. The third variable, for example, is C and is described in the third field of Word 1. The number 7 points to C in the symbol table, the number 4 points to the fourth row in IFNCRA, and the number 1 indicates that C is the first argument of the reference.

EXAMPLE 2

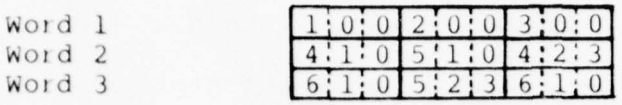
```
WRITE (IO,10) A,B,((C(I,J),I=1,N),J=1,N)
```

Given the above WRITE statement, the following arrays are constructed:

SYMBOL TABLE

1	A
2	B
3	C
4	I
5	J
6	N

IARGS (for CDC)



Information on the WRITE statement is stored in the IARGS array. Assume that the variables encountered are stored in the symbol table as in the previous table. There are nine occurrences of variables in the I/O list that are stored in the IARGS array. Each word contains three fields, each field describing a different variable. The second field of Word 3, for example, describes the second occurrence of the variable J. The number 5 points to the variable J in the symbol table, the number 2 indicates that J is an implied DO index, and the number 3 points to the symbol table location of the first variable (C) of the implied DO.

2.2.96 COMMON Block GIRL.

1. NTERMS - Number of terminal symbols in the grammar. Presently set equal to 19.

2-20. PLUS - OPRAND - The 19 terminal symbols used by the parser. OPRAND signifies a constant or a variable.

2.2.97 COMMON Block GLOBAL.

1. NBLK - The number of entries in the COMMON block table BLKTBL(200).

2. NREF - The number of entries in EXTTBL(100).

3. NSUBS - The number of entries in ISUBS(100).

4. BLKTBL(200) - This array contains a list of all COMMON blocks encountered in the program unit(s) being analyzed. The Interface Definition file location of each COMMON block is stored rather than the name of the block.

5. EXTTBL(100) - This array contains a list of all functions and subroutines which were referenced by the program unit(s) being analyzed. The Interface Definition file location of each function or subroutine is stored.

6. ISUBS(100) - This array contains a list of the program unit(s) being analyzed. The Interface Definition file location of each program unit is stored.

2.2.98 COMMON Block HL.

1. HOL - GIRL value for HOL link in GIRL structure.

2. ACTION - GIRL value for ACTION link in the syntax graph.

3. FUNC1 - GIRL value that indicates truncation of COMPLEX function.

4. FUNC2 - GIRL value that indicates truncation of DOUBLE PRECISION function.

5. FUNC3 - GIRL value that indicates truncation of REAL function.

6. LEFT - GIRL value that holds inserted left parenthesis of truncation function.

7. RIGHT - GIRL value that holds inserted right parenthesis of truncation function.

8. STRING - GIRL value that produces GIRL string structure.

9. MAXJ - Largest index of successfully parsed substring.

2.2.99 COMMON Block INPOUT.

1. NCALL - The number of statements in the program unit being analyzed; set by subroutine BUILD.

2. IN - The logical unit number of the input device to be used for the program unit(s) being analyzed. This number is input by the user and is 5 for cards and 7 for tape or disk.

3. IOP - The logical unit number of the output device for the revised program unit (set to 8).

2.2.100 COMMON Block JL.

1. JSTOP - If a level exists, JSTOP>0. If a level does not exist, JSTOP=0.

2.2.101 COMMON Block LABELS. This block contains information about statement labels.

1. STATRA(2,200) - This array is the statement number table, created anew for each program unit. As many as 200 statement numbers are stored, each statement number being described by two words.

Word 1 - Contains the statement number.
Word 2 - Contains six fields of information
field 1 - statement type, according to Figure 2-04.
field 2 - 1 if there has been a statement with this label and 0 otherwise.
field 3 - 1 if the statement label has been referenced and 0 otherwise.
field 4 - 1 if the label is a DO terminal and 0 if it is not.
field 5 - 1 if the label is on a complete path and 0 if it is not.
field 6 - points to the basic block containing the label.

Fields 1 through 5 have the following bit configurations, respectively: 1-6, 7-9, 10-12, 13-15, and 16-18. For the IBM 360, field 6 is bits 19-32; and for the UNIVAC 1108 and CDC, field 6 is bits 19-36.

2. NLABEL - The integer value of the number of statement numbers in the statement number table.

2.2.102 COMMON Block LIST. This block stores the information that resides in the Interface Definition file. The ISUBLT array lists the symbolic names of program units and labeled COMMON blocks and stores information on those names. The INTFAC array stores information about arguments and Category 1 COMMON elements.

1. NLIST - The number of entries in the ISUBLT array.

2. NINTFC - The number of entries in the INTFAC array.

3a. ISUBLT(2,200) - This array is the CDC version of the ISUBLT array. It stores as many as 200 symbolic names, each name being described by two words.
Word 1 - Contains the symbolic name.
Word 2 - Information about that name.
Bits 1-6 - (For subprograms) The number of arguments.
(For Category 1 COMMON blocks) The number of groups (a group is a block of variables of the same type).
(For Category 2 and 3 COMMON blocks) A value of zero.
Bits 7-10 - (For a program unit). One of the following values:
0 - User supplied (not in the Interface Definition file)
1 - Subroutine module

2 - Function module
3 - Ancillary subprogram
4 - ANSI function
5 - Main program
6 - Extraordinary subroutine
(For labeled COMMON blocks) One of the following values:
7 - Category 1 COMMON block
8 - Category 2 COMMON block
9 - Category 3 COMMON block

Bits 11-13 - Function type (1 for real, 2 for complex, 3 for double precision, 4 for integer, 5 for logical). If the symbolic name is a COMMON block, a zero is stored.

Bit 14 - Set to 1 if a subprogram has a variable number of arguments and 0 otherwise.

Bits 15-30 - Maximum size of blank COMMON (if any).

Bits 31-60 - Points to the beginning location in the INTFAC array, where information on the arguments or Category 1 COMMON is described.

3b. ISUBLT(3,200) - This array is the UNIVAC 1108 version of the ISUBLT array. Three words are used to describe a symbolic name. The first two words are the same as in 3a, except that Bits 15-36 store the size of blank COMMON. Word 3 contains the pointer into the INTFAC array.

3c. ISUBLT(4,200) - This array is the IBM 360 version of the ISUBLT array. Four words are used to describe a symbolic name. Words 1 and 2 contain the symbolic name. Word 3 is the same as Word 2 of the CDC version, except that Bits 15-32 store the size of blank COMMON. Word 4 contains the pointer into the INTFAC array.

4a. *INTFAC(300) - This array is the CDC version of the* INTFAC array. It contains information about subroutine argument lists and Category 1 COMMON blocks.

For subprograms, arguments are stored six fields per word, each field of the following form.

Bits 1-3 - Argument type (1 for real, 2 for complex, 3 for double precision, 4 for integer, 5 for logical)

Bits 4-6 - Argument dimensionality - (0,1,2, or 3 for 0,1,2, or 3 subscripts)

Bits 7-9 - Argument I/O status (0 for output, 1 for input and output, 2 for input)

For Category 1 COMMON blocks, groups are stored three fields per word, each field of the following form:
Bits 1-17 - Size of group
Bits 18-20 - Type of group (1 for real, 2 for complex, 3 for double precision, 4 for integer, 5 for logical).

4b. INTFAC(500) - This array is the UNIVAC 1108 version of the INTFAC array.

For subprograms, arguments are stored four fields per word, each field of the following form:
Bits 1-3 - Argument type
Bits 4-6 - Argument dimensionality
Bits 7-8 - Argument I/O status
Bit 9 - Set to 1 if the argument is in an expression and 0 if not (For CDC version, this value is stored in the IFNCRA array).

For Category 1 COMMON blocks, groups are stored in two fields per word, each field of the following form:
Bits 1-15 - Size of group
Bits 16-18 - Type of group

4c. INTFAC(600) - This array is the IBM 360 version of the INTFAC array.

For subprograms, arguments are stored three fields per word, each field of the same form as in 4b.

For Category 1 COMMON blocks, groups are stored one field per word, each field of the same form as in 4b.

2.2.103 COMMON Block LOGIC.

1. LOG - Set to 1 if a logical operator or constant is found and 0 otherwise.

2. LOGST - Location in input string where search for logical operator or constant is to start.

2.2.104 COMMON Block LVARGS.

1. IFUNC - The link of the triple, also known as the function. It must be a random number as defined by LVGRN.

2. IARG - The source node of the triple, also known as the argument of the function. It must be a random number as defined by LVGRN.

3. IADD - The location (address) of the triple in the buffer.

4. IPOS - The position from the top (if positive) or bottom (if negative) of the multi-value list into which the new value will be placed. If ITYP is specified, only that type of value is counted when determining the position in the list.

5. ITYP - The type of value to be searched for in a multi-value list. It is used for destructive and nondestructive insertions and may have the following values:
   - -1 - Delete the entire function
   - 0 - Random number
   - 1 - Integer data
   - 2 - Hollerith data
   - 3 - No specified type

6. IVAL - The retrieved value. However, if the value cannot be found, IVAL is set to -1.

7. LSTHED - The location of the head of the multi-value list, otherwise,
   - 0 - if no list is found
   - -1 - if a single valued list is found

8. NVAL - The number of values (up to ten) to be inserted in the graph.

9. IDSTRY - Indicates the type of insertion to be made according to the following values:
   - 0 - Normal insertion - the triple is always placed at the end of the (null) list.
   - 1 - Destructive insertion - the IPOSth member of the ITYPth type from the top or bottom of a list (depending on the sign of IPOS) is replaced by the contents of IVALS(1).
   - 2 - Nondestructive insertion - the contents of IVALS(1) are wedged into the list, making the new value the IPOSth member of the ITYPth type from the top or bottom of the list (depending on the sign of IPOS).

10. IVALS(10) - The values or sink nodes to be inserted; they may be any of the following types:

. Random number as defined by LVGRN
. Integer data (up to 2**16)
. Hollerith data

11. ITYP1(10) - A type description for each value in IVALS(i) to be inserted according to the following values:
    0 - Random number
    1 - Integer data
    2 - Continuing Hollerith data
    3 - The only or final cell of a Hollerith data string

12. NSKIP - Saved index defeat switch. If NSKIP=1, the saved index operation is skipped. Otherwise the saved index is in effect.

2.2.105 COMMON Block LVFLAG. This COMMON block contains integers which are used as masks to extract and insert information from COMMON block LVVTR4.

2.2.106 COMMON Block LVRAND. This COMMON block is used in the computation of random numbers for the nodes on the graph.

2.2.107 COMMON Blocks LVTABL and LVVSEQ. These COMMON blocks are generated by the GIRL software and are not used by AUDIT.

2.2.108 COMMON Block LVVTR1.

1. MEMSZE - GIRS memory size (set to 1000). This value is set at the time the graph is created and can only be changed by recreating the graph.

2. REGASP - If a location for a triple is computed and that location is already in use, this value points to the next available location where the triple may be stored.

3. NODSPC(1000) - For each triple (there may be as many as 1000), one of three possible items is stored in each location:
    a. Source node of a single-value list
    b. Source node of a multi-value list
    c. Sink node of a value on a multi-value list.

2.2.109 COMMON Block LVVTR2.

1. LSTSPC(1000) - For each triple (there may be as many as 1000), the sink node of a single-value list is stored. If the list is a multi-value list, the pointer to the next item in the list is stored. The head of the list points to the first sink node, and

the last sink node points back to the head of the list; thus a circular structure is created.

2.2.110 COMMON Block LVVTR3.

1. LNKSPC(1000) - For each triple (of as many as 1000), the conflict list pointer is stored. All heads of lists which hash to the same location are linked together on a circular list. If an entry has no conflicts it points to itself.

2.2.111 COMMON Block LVVTR4.

1. FLGSPC(1000) - For each triple (of as many as 1000), five flags are stored:
   - Bit 1 - 1 if head of a multi-value list
     0 if not
   - Bit 2 - 1 if cell is in working space
     0 if cell is available
   - Bit 3 - 1 if value is on a multi-value list
     0 if not
   - Bit 4-5 - left blank
   - Bit 6 - 1 if head of conflict list
     0 if not
   - Bit 7-8 - 0 if random number
     1 if integer data
     2 - if continuing Hollerith string
     3 - if only or final Hollerith string

2.2.112 COMMON Blocks LVVTR5, LVVTR6, LVVTR7, and LVVTR8.
These COMMON blocks are generated by the GIRL software and are not used by AUDIT.

2.2.113 COMMON Block NEED.

1. START - Beginning state.
2. ASSOC - GIRL pseudo-random variable for associate.
3. LEVEL - GIRL pseudo-random variable for level.
4. STOP - GIRL pseudo-random variable for a permissable final state. Also a stop link to a stop state is permissable.

2.2.114 COMMON Block NEEDS.

1. STJ - The value of the token (in input string) presently being examined.
2. JSTACK - Index which points to a location in the stack.

3. R - Present state in the graph.

4. JAS - Index of last associate taken.

5. J - Index of present token in the input string.

6. JLAST - Index to the beginning of a primary, which is a parenthesized expression.

7. RTEMP - Value of last state of the graph, in case of immediate need to back up.

8. STACK(400) - Array which holds all the nodes visited as well as other information which is necessary in order to back up (i.e., recognize a context free language).

2.2.115 COMMON Block NOPAR.

1. NOPAR - Stack variable used to keep track of recursive functions.

2. NDEP - Same as NOPAR.

3. NDEPTH - Function counter.

4. NFLAG - Stack variable for use in I/O list processing.

2.2.116 COMMON Block NTIMES.

1. NTIMES - Flag to indicate that the syntax graph has been read from disk to core.

2. I - If the syntax graph runs out of storage space while parsing a statement, I indicates how far the statement has been parsed.

2.2.117 COMMON Block REALNO.

1. IREAL - Set to 1 if a real number is found and 0 otherwise.

2. IRELND - The last location of a character string representing a real number.

3. IP - If a character string is to be tested to see if it is a real number, this value points to the beginning location of the string.

2.2.118 COMMON Block STFUNC.

1. NSTFNC - The number of entries in the ISTFNC array.

2. ISTFNC(10) - This array contains symbol table locations of all statement functions in the program unit.

2.2.119 COMMON Block STRING.

1. NTYPE - Indicates the type of an expression according to the following values:
   1 - Arithmetic
   2 - Logical
   3 - I/O list

2. NSTR - The number of language elements in STR.

3. STR(500) - Encoded expression for a character string (see SUBROUTINE EXPR).

2.2.120 COMMON Block TYP.

1. NARRAY - The number of subscripts in an array.

2. TYPE1 - The type of the present oprand of the present operator.

3. TYPE2 - The type of the first oprand of a binary operator. TYPE1 will hold the type of the second oprand.

4. ERRFLG - The value of .TRUE. if a syntactic error has occurred. The value of .FALSE., otherwise.

2.2.121 COMMON Block VAR.

1. VFOR(15) - This array contains the display code for output of an incorrect input string.

2. NUMCHR - The number of characters presently placed in the last used word of VFOR.

3. NCHRP - The final number of tokens (including inserted functions and parens) in the transformed input string.

4. CHAR - The display code of input characters.

5. NDICT - The equivalent value for input characters. ($-1 \geq NDICT \geq -18$)

2.2.122 COMMON Block WASTE. This is a dummy COMMON block used to store local arrays.

SECTION 3. INPUT/OUTPUT DESCRIPTIONS

3.1 General Description. The inputs and outputs of the AUDIT system are of three types:

1. Card, tape, and/or disk input furnished by the user.
2. Disk input and output which is of no direct concern to the user.
3. Printed output that is of direct concern to the user.

The inputs and outputs of the AUDIT system that fall into these three categories are described in the following section.

3.2 Characteristics, Organization, and Detailed Description of System Data.

3.2.1 User Input. Certain items must be furnished by the user each time the AUDIT system is executed:

1. An options card
2. The software to be examined
3. The Interface Definition file
4. The syntax graph file
5. A package of variable precision functions for each truncation operator (bit configuration) desired. (Needed only if the variable precision mode is selected.)

3.2.1.1 Options Card. The first card in the user's input deck is the options card, always supplied as card input. If the software to be examined is on tape or disk and the variable precision mode has not been selected, the options card will be the only card input. The options card is punched in the format (I1,3X,I1,3X,I1,3X,I1) and is constructed as follows:

1. Column 1 indicates the mode of operation:
   - 1 - Audit mode only, or both the audit mode and the variable precision mode.
   - 2 - Audit mode and roll call mode (subprogram to be examined does not contain a reference to utility module ROLCOL).
   - 3 - Audit mode and roll call mode (subprogram to be examined contains a reference to utility module ROLCOL).

2. Column 5 indicates the input logical unit number for the software to be examined:
   - 5 - Software is on cards
   - 7 - Software is on tape or disk (For CDC and UNIVAC 1108)
   - 2 - Software is on tape or disk (For IBM 360)

3. Column 9 indicates whether or not the flow analysis mode is to be used:
   0 - No flow analysis
   1 - Moderate flow analysis
   2 - Comprehensive flow analysis

4. Column 13 indicates whether or not intrinsic and basic external function names are to be checked for misuse (See SESCOMPSPEC3, Section 8):
   0 - Do not check names
   1 - Check names

The AUDIT system always performs the audit mode. The variable precision or roll call mode may also be selected (Column 1). A value of 1 in Column 1 indicates one of two possible actions: (1) audit mode only, or (2) audit mode and variable precision mode. The AUDIT system distinguishes between these two options by the additional control cards and input cards for the variable precision mode. An entire executable program (main program, all subprograms, and data) must be submitted when the variable precision mode is selected. No additional software (other than the one executable program) may be examined by the variable precision mode. The roll call mode is only used to examine the root program unit of a module. Main programs, ancillary subprograms, or extraordinary subroutines should not be examined by the roll call mode.

3.2.1.2 Software To Be Examined. The software to be examined is input to the AUDIT system; it must be on cards, tape, or disk, and it may be on only one of the three mediums. If the software is on cards, it must immediately follow the options card, and a value of 5 must be punched in Column 5 of the options card. If the software is on tape or disk (must be in source format), the tape or disk involved must be identified as logical unit number 7 or 2, and a value of 7 or 2 must be punched in Column 5 of the options card.

If the variable precision mode is selected, only one executable program (main program, subprograms, and data) may be input. If the roll call mode is selected, only the root program unit of a module may be input. Only one main program may be input per execution. There is a limit to the number of subprograms that may be input per execution. This limit depends on the complexity of the subprograms. A reasonable upper limit on the number of subprograms input per execution is 10.

3.2.1.3 Interface Definition File. Before any software can be examined, the Interface Definition file must be constructed by auxiliary program SESLIST (see Section 4.4.1). The Interface Definition file is an unformatted file which is read from logical unit 4 by AUDIT's main program and is stored in arrays

ISUBLT and INTFAC of COMMON block LIST. The file contains information about all program units and COMMON blocks that may be referenced by the software being examined.

For root program units of modules, main programs, ancillary subprograms, extraordinary subroutines, intrinsic and basic external functions, and other external subprograms, the Interface Definition file contains information on the argument list (if any) of the program unit and other appropriate information. The file also contains information about all blank COMMON and labeled COMMON blocks.

The Interface Definition file consists of two parts: (1) the basic Interface Definition file, and (2) the user supplied Interface Definition file. The basic Interface Definition file consists of information on the intrinsic and basic external functions, and the labeled COMMON block SESCOM. This basic file (listed in Appendices B, C, and D) is considered to contain information that may be referenced by all software produced for the SESCOMP system. Each time new software is to be examined, the user supplied file is constructed for the specific software. The user supplied file should not contain any information that is present in the basic file. This user supplied file is then added to the basic file to make up the Interface Definition file for the software to be examined. A maximum of 200 symbolic names may be entered in the file.

As stated before, the Interface Definition file is created by the independent program SESLIST. The information that constructs the file is punched on cards. SESLIST reads the cards, packs the card images, and writes them out unformatted onto logical unit 4. This file is saved and used as an input file for the AUDIT system. The user must rerun SESLIST each time a change in the file is necessary. The old file is discarded and the new one is saved. While operating under the audit mode, each reference to a program unit or labeled COMMON block is checked for consistency against the Interface Definition file. If a program unit or labeled COMMON block is not in the file, the name and its associated information is added to the ISUBLT and INTFAC arrays (COMMON block LIST). Any subsequent reference to that program unit or labeled COMMON block is then checked for consistency with the first reference.

The user supplied file consists of each reserved symbolic name used by the software to be examined (excluding those names already in the basic file). There will be at least one card of information associated with each name. For subroutine and function subprograms with arguments and Category 1 labeled COMMON blocks, additional cards will be needed. Subroutine and function subprograms without arguments,

Category 2 and 3 labeled COMMON blocks, or blank COMMON will each have only one card associated with it.

SESLIST requires at least one input card for each subprogram and main program. This card is constructed as follows:

Columns 1-6	The symbolic name (left justified)
Columns 8-9	The number of arguments (right justified); (if none, punch a zero in Column 9)
Column 11	Function type (if the program unit is a function subprogram) 1 - Real 2 - Complex 3 - Double Precision 4 - Integer 5 - Logical (If the program unit is not a function subprogram, punch a zero)
Column 13	The class of the symbolic name: 1 - Subroutine module 2 - Function module 3 - Ancillary subprogram 4 - Intrinsic or basic external function 5 - Main program 6 - Extraordinary subroutine
Columns 15-20	The size of blank COMMON, if any (right justified)

If the program unit doesn't have any arguments, only the above card is needed to describe the symbolic name. Otherwise, more cards are needed to describe the arguments. Each argument requires a 4-column field, with as many as 20 fields per card. If there are more than 20 arguments, a second card is required; if there are more than 40 arguments, a third card is required, and so on. A maximum of 63 arguments is permitted. The fields are ordered left to right in the same order as the arguments. A field is constructed as follows:

Column 1	The argument type: 1 - Real 2 - Complex 3 - Double Precision 4 - Integer 5 - Logical
Column 2	The dimensionality of the argument: 0 - Non-subscripted simple variable 1 - Single-subscripted array 2 - Double-subscripted array 3 - Triple-subscripted array
Column 3	Argument I/O status: 0 - Output

1 - Input and output
2 - Input

Column 4 — Blank (serves as a separator between fields)

For example, utility module SESPL1 is described by the following two cards:

(Card 1) SESPL1 3 0 1
(Card 2) 402 402 402

SESLIST requires at least one input card for each labeled COMMON block. This card is constructed as follows:

Columns 1-6 — The symbolic name (left justified)

Column 9 — (For Category 1) The number of groups contained in the block, a group being a set of consecutive words of the same type.
(For Category 2 and 3) Left blank

Column 11 — Always zero

Column 13 — The category of labeled COMMON:
7 - Category 1 labeled COMMON block
8 - Category 2 labeled COMMON block
9 - Category 3 labeled COMMON block

Columns 15-20 — (For Category 1) The size of the block (right justified)
(For Category 2 and 3) Left blank

Category 2 and 3 labeled COMMON blocks do not require additional cards; Category 1 labeled COMMON blocks require additional cards describing the structure of the blocks. The additional Category 1 cards are constructed as follows:

Each group, a set of consecutive words of the same type, requires a field of eight columns. As many as ten 8-column fields may be punched per card, with additional cards used as needed. The fields are ordered left to right in the same order as the groups. Each 8-column field is constructed as follows:

Columns 1-6 — Number of words in the group (right justified). A double precision or complex variable is counted as two words.

Column 8 — Type of the group:
0 - Hollerith
1 - Real
2 - Complex
3 - Double precision
4 - Integer
5 - Logical

For example, the description of the Category 1 labeled COMMON block SESCOM is punched on two cards, as follows:

```
(Card 1)      SESCOM  2   7     25
(Card 2)          13 0    12 4
```

When organizing the input cards for SESLIST, the card(s) that describes the arguments or the groups must immediately follow the symbolic name the card(s) is associated with. Although there is no prescribed order for the symbolic names, an alphanumeric arrangement is usually convenient. Such an arrangement simplifies the retrieval of a reserved name by the user. The file that is constructed by program SESLIST must be saved (catalogued) as a tape or disk file. Each time the AUDIT system is executed, this Interface Definition file must be input to the system.

3.2.1.4 Syntax Graph. The syntax graph is a plex data structure which contains the syntax for all arithmetic and logical expressions and I/O lists. It is constructed by program GRAPH, which reads the graph from punched cards, and then packs and writes out the information as an unformatted file onto logical unit 4. Whenever one of the expressions or lists is encountered by the AUDIT system, the structure of the expression or list is checked against the graph for validity. The graph is an unformatted file which is read by the parser subprograms from logical unit 19 and is stored in COMMON blocks GIRL, LVTABL, LVVSEQ, LVVTR1, LVVTR2, LVVTR3, and LVVTR4. The syntax graph file remains constant. It is constructed just once and is then reused whenever the AUDIT system is executed. The user need not concern himself with the data to construct the file. The data has already been constructed and will not change. The user need only create the file by running program GRAPH. He then saves the file as a tape or disk file.

3.2.1.5 Variable Precision Functions. When AUDIT is operating under the variable precision mode, a package of variable precision functions must be input for each bit configuration desired. The following three function subprograms (a package) are needed for each bit configuration:

1. REAL FUNCTION Q1REAL
2. DOUBLE PRECISION FUNCTION Q1DPRE
3. COMPLEX FUNCTION Q1COMP

The user may simulate word lengths of 30 to 40 bits for the CDC 6000, 24 to 31 bits for the IBM 360, and 30 to 35 bits for the UNIVAC 1108. Each processor uses a unique set of packages.

3.2.2 AUDIT-Generated I/O. The AUDIT system generates several temporary disk files which are used at various stages of the AUDIT execution. These files are of no direct concern to the user.

3.2.2.1 Revised Program Files. Each time the audit mode is executed, the software being examined is revised and the revised program resides on logical unit 8. There are two different kinds of revised program files generated.

1. If the roll call mode is not selected, all arithmetic operations (+,-,/,*,**) that involve real, complex, and double precision variables or constants are changed into references to the variable precision function subprograms Q1REAL, Q1COMP, and Q1DPRE. For example, A=B+C is transformed into A=Q1REAL(B+C), assuming A, B, and C are real. If the variable precision mode is selected, the revised program file is used to perform the variable precision calculations. Otherwise, the file is not used.
2. If the roll call mode is selected, all references to function and subroutine modules are changed to references to subroutine ROLCHK, and all READ statements are omitted. These revisions are performed for a single subprogram. The main program ROLCAL is added to the file. This file is then used to execute the roll call mode.

3.2.2.2 AUDIT Module List. The AUDIT module list is generated only if the roll call mode is selected. The list consists of function and subroutine module references encountered during the audit mode. The list is contained on an unformatted file which resides on logical unit 9. It is used by subroutine CMPARE to verify that all of the referenced modules were also referenced in the SESCOMP roll call mode.

3.2.2.3 ROLCHK Module List. The ROLCHK module list is generated during execution of the roll call mode. The list consists of all modules referenced in the SESCOMP roll call mode. It is an unformatted file which resides on logical unit 3. This file is constructed by subroutine ROLCHK. It is used by subroutine CMPARE to compare with the AUDIT module list to insure that the same names are on both lists.

3.2.3 Roll Call Output Files. The roll call output files contain the contents of the SESCOMP output devices X, Y, and Z, which are created by the simulated execution of the program unit being examined. The contents of output devices X, Y, and Z are written on logical units 13, 14, and 15, respectively. These devices contain the Module Identification Fields, use counts, and buffer tracing information pertinent to the simulated SESCOMP roll call modes. The user may display the contents of these files at the completion of execution, assuming the roll call mode has been selected.

3.2.4 Printed Output. Each program unit of the software being examined has its own printed output. After the printout

associated with all program units has been generated, a global reference table is printed which contains information on all the program units processed. A description of the printout associated with each program unit follows.

AUDIT prints each statement of the software being processed together with a diagnostic message if any intrastatement errors have been found. Each assignment or CALL statement that contains a real, double precision, or complex operation is transformed to include the references to Q1REAL, Q1DPRE, and Q1COMP, regardless of whether or not the variable precision option was selected. If the roll call mode has been selected, each module reference is transformed into a reference to subroutine ROLCHK, and all READ statements are made into comment statements. All non-intrastatement errors are printed ahead of and following the symbol table for the program unit. The symbol table lists all symbolic names and the type, dimensionality, and relocation of each name. In addition, all externals, statement labels, and COMMON blocks are listed, along with associated information. If the flow analysis mode is selected, the results of the flow analysis are printed. For each variable that is referenced but not defined along some path, the variable and the statement labels of the path are indicated. A variable may be flagged only once. The number of paths checked are also indicated.

Each time AUDIT is executed, a global reference table is printed. This table lists all external references, labeled COMMON blocks, and program units encountered (and their associated information) for all the program units examined. If the roll call mode is selected, the following items will be printed after the global reference table for each program unit.

1. Compilation of the revised program unit (see Section 3.2.2.1).
2. Compilation of main program ROLCAL.
3. Load map.
4. Results of the roll call check.
5. Whatever the program unit has written on output devices X, Y, and Z.

If the variable precision mode is selected, the revised program file will be compiled. For each bit configuration selected, the (printed) output of the executable program is printed. Appendix A contains a representative sample of the various kinds of printouts.

SECTION 4. PROGRAM ASSEMBLING, LOADING, AND MAINTENANCE PROCEDURES

4.1 Input/Output Requirements. The following components make up the AUDIT system:

1. Audit mode source code software
2. Roll call mode source code software
3. Program SESLIST and data
4. Program GRAPH and data
5. Variable precision function subprograms.

For the purposes of this document it is assumed that the audit and roll call mode source code software is stored on tape as two source files, and that the other components exist in the form of card decks. The roll call source software consists of subroutines CMPARE, MODID, and ROLCHK. The audit mode source software consists of all the programs described in Section 2.2 (excluding QlCOMP, QlDPRE, QlREAL, ROLCAL, CMPARE, MODID, and ROLCHK). Before the AUDIT system can be executed, programs SESLIST and GRAPH must create the Interface Definition file and the syntax graph file, respectively. The user then specifies these two files as input to the audit mode software and inputs the options card (must be card input) and the software to be examined (must be card, tape, or disk input). If the variable precision mode is selected, the variable precision function subprograms (for the bit configurations desired) are then input. The control cards are all that remain to execute the AUDIT system.

4.2 Procedures. The AUDIT system is executed in the batch mode. As stated in Section 4.1, it is assumed that the audit and roll call mode source software is stored on tape as two consecutive files and that all other components are in the form of card decks. This need not be the case, but for the purposes of explaining all possible operating procedures, this construct is assumed. It is also assumed that all permanent files generated by the AUDIT components are stored as disk files. It will be easy for the user to adapt to other physical mediums if he shall so choose.

The control cards needed to execute all possible options of the AUDIT system will be listed and described for the following three processors:

1) CDC 6000 series, SCOPE 3.4 operating system
2) UNIVAC 1108, EXEC-8 operating system
3) IBM 360, OS-360 operating system.

For each possible option, only Column 1 and Column 5 of the options data card are discussed. The flow analysis mode may be selected for any audit mode, roll call mode, or variable precision mode, and is controlled by Column 9 of the options card. The ANSI name check may also be selected for any mode, and is controlled by Column 13. If the variable precision mode is selected and the executable program requires tape or

disk data, such data must be made available to the program by assigning the appropriate input files. The files must be rewound for each bit configuration executed.

It is difficult to state precise timing estimates of the various AUDIT options. It is best to first analyze a few subprograms using the various options and then use those times as a base for future runs. The only option that may be time consuming is the flow analysis mode, where the time is dependent on the number of paths. For subprograms that have paths that number in the thousands, it may be better to use the moderate flow analysis. Only by trial and error will a user be able to get a feeling for the amount of time needed to execute. When running the variable precision mode, the user should have an estimate of the execution time of the program to be analyzed. Then for all the bit configurations to be analyzed, the user should allow enough time to execute the program the desired number of times (in addition to the audit time). Timing estimates were computed for some of the programs listed in Appendix A. To perform the audit mode for the main program SAMP, the extraordinary subroutine INOUT, and the BLOCK DATA subprogram, it took about 5.0 CPU seconds on the CDC 6400 and the UNIVAC 1108 and 2.5 seconds on the IBM 360. To perform the roll call mode for subroutine APNDGA, it took about 20.0 CPU seconds for the CDC 6400 and the UNIVAC 1108 and 14.5 seconds for the IBM 360.

4.2.1 CDC 6000. The control cards listed in this section are applicable to the SCOPE 3.4 operating system. Appropriate modifications are easily made for different operating systems.

4.2.1.1 Initial File Creation. The following four files must be created initially:

1. Interface Definition file
2. Syntax graph file
3. Absolute binary version of the audit mode source software
4. Relocatable binary version of the roll call mode source software.

To create the Interface Definition file, execute program SESLIST and catalogue the output file with a permanent file name of IDFILE. To create the syntax graph file, execute program GRAPH and catalogue the output file with a permanent file name of SYNTAXGRAPH. The audit binary version (PFN=AUDITBIN) and the roll call binary version (PFN=ROLLCALLBIN) are created from the source labeled tape (L=AUDITSOURCE) using the following control cards:

1. Standard SCOPE Job card (requesting CM130000, MT1, and T300)
2. Standard SCOPE charge card
3. VSN(TAPE=SES01=SLOTXX)

```
 4. LABEL(TAPE,L=AUDITSOURCE,R) (NORING/SES01/SLOTXX)
 5. REQUEST(AUD,*PF)
 6. REQUEST(RLCL,*PF)
 7. COPYBF(TAPE,AUDIT)
 8. COPYBF(TAPE,ROLL)
 9. RETURN(TAPE)
10. REWIND(AUDIT,ROLL)
11. RFL(70000)
12. FTN(I=AUDIT,B=AUDBIN,L=0,OPT=2)
13. FTN(I=ROLL,B=RLCL,L=0,OPT=2)
14. CATALOG(RLCL,ROLLCALLBIN,ID=XXXX,AC=XXXXXXXXXX)
15. RFL(130000)
16. LOAD(AUDBIN)
17. NOGO(AUD)
18. CATALOG(AUD,AUDITBIN,ID=XXX,AC=XXXXXXXXXX)
19. 6/7/8/9 End-of-file
```

EXPLANATION OF CARDS

3.-4. Mount AUDIT source tape. Note that these cards may be installation dependent.
5.-6. Request permanent file devices for files.
7.-9. Copy source files from tape to temporary disk files and release tape.
12.-13. Compile audit and roll call source. Note that if a listing of the source is desired, omit L=0.
14. Catalog the roll call mode programs in relocatable binary form.
16. Load the relocatable binary file.
18. Catalog the audit programs in absolute binary form.

4.2.1.2 Execute Audit Mode from Cards. If the software to be examined is on cards, the following sequence of cards is used to execute the audit mode.

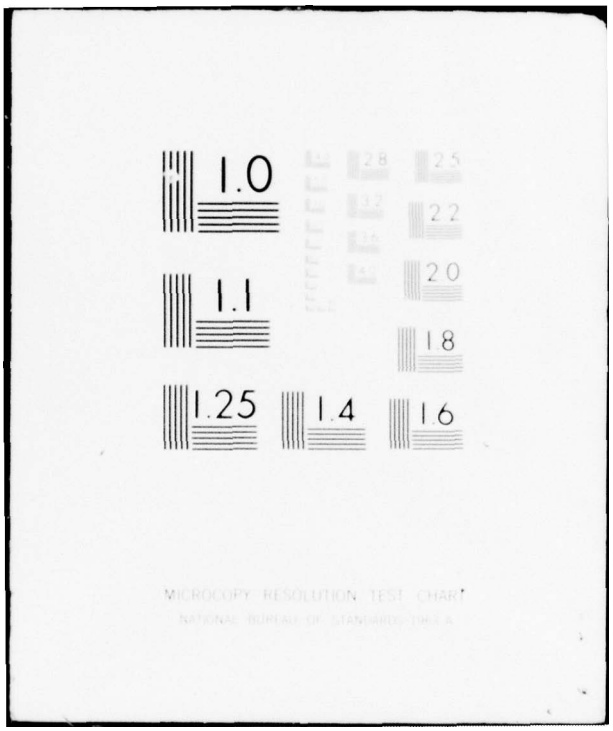
```
 1. Standard SCOPE Job card (CM130000)
 2. Standard SCOPE charge card
 3. ATTACH(TAPE4,IDFILE,ID=XXXX)
 4. ATTACH(TAPE19,SYNTAXGRAPH,ID=XXXX)
 5. ATTACH(AUD,AUDITBIN,ID=XXXX)
 6. AUD.
 7. 7/8/9 End-of-record
 8. Options card
 9. Software to be examined
      .
      .
      .
10. 6/7/8/9 End-of-file
```

AD-A043 922 DAVID W TAYLOR NAVAL SHIP RESEARCH AND DEVELOPMENT CE--ETC F/G 9/2
MAINTENANCE MANUAL FOR AUDIT, A SYSTEM FOR ANALYZING SESCOMP SO--ETC(U)
AUG 77 R J WYBRANIEC, R REGEN

UNCLASSIFIED DTNSRDC-77-0075-VOL-1 NL

2 OF 2
AD A043922

END
DATE FILMED
9 -77
DDC


1.0
1.1
1.25
1.4
1.6
2.8
2.5
2.2
2.0
1.8
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963 A

EXPLANATION OF CARDS

3.-4. Attach input files.
5.-6. Attach and execute audit absolute binary file.
8. Options card is punched with a 1 in Column 1 and a 5 in Column 5.
9. At least one program unit is needed. Only one main program is permitted.

4.2.1.3 Execute Audit Mode from Sequential Disk or Tape File. A sequential file must contain at least one program unit. If there is more than one program unit, all program units must be consecutively stored without any embedded end-of-records between program units. Only one main program is permitted. If the software to be examined is on a sequential disk file, the following sequence of cards is used to execute the audit mode.

1.-5. Same as 1-5 of Section 4.2.1.2
6. ATTACH(TAPE7, PFN, ID=XXXX)
7. AUD.
8. 7/8/9 End-of-record
9. Options card
10. 6/7/8/9 End-of-file

EXPLANATION OF CARDS

6. PFN is the name of the permanent file on which the software to be examined resides.
9. Options card is punched with a 1 in Column 1 and a 7 in Column 5.

If the software to be examined is on a sequential standard SCOPE labeled tape file, the following two cards replace Card 6.

6a. VSN(TAPE7=VRNO)
6b. LABEL(TAPE7,L= NAME ,R) (VRNO/NORING)

where VRNO is the visual reel number of the tape, and NAME is the tape label name. If the software to be examined is on a sequential stranger tape, the following two cards replace Card 6.

6a. VSN(TAPE7=VRNO)
6b. REQUEST(TAPE7,DEN,TYPE) (VRNO/NORING)

where DEN is HI, HY, or HD (556 seven track, 800 seven track, and 800 nine track, respectively) and TYPE is S or L (S for record size $\leq$512 words and L for record size >512 words).

4.2.1.4 Execute Audit Mode from Update Disk or Tape File. The update file must contain at least one program unit. Only one main program is permitted per execution. If the software to be examined is stored on an update disk file, the following sequence of cards is used to execute the audit mode.

1.-5. Same as 1-5 of Section 4.2.1.2.
6. ATTACH(OLDPL, PFN ,ID=XXXX)
7. UPDATE(Q,C=TAPE7)
8. AUD.
9. 7/8/9 End-of-record
10. *C DECK1, DECK2, ..., DECKn
11. 7/8/9
12. Options card
13. 6/7/8/9 End-of-file

EXPLANATION OF CARDS

6. PFN is the name of the permanent file on which the software to be examined resides.
7. Call the update utility to select the decks from the update file that the user wishes to examine. If the user wishes to examine all decks on the update file, replace the Q with an F.
10. DECK1, DECK2, ..., DECKn are the specific decks to be examined. If an F is on the update card (Card 7), omit this card.
12. Options card is punched with a 1 in Column 1 and a 7 in Column 5.

If the software to be examined is on an update labeled tape file, the following two cards replace Card 6.

6a. VSN(TAPE7=VRNO)
6b. LABEL(OLDPL, L= NAME , R) (VRNO/NORING)

where VRNO is the visual reel number of the tape and NAME is the tape label name. If the software to be examined is on an update stranger tape file, the following two cards replace Card 6.

6a. VSN(TAPE7=VRNO)
6b. REQUEST(OLDPL,DEN,TYPE) (VRNO/NORING)

where DEN is HI, HY, or HD (556 seven track, 800 seven track, and 800 nine track, respectively) and TYPE is S or L (S for record size $\leq$512 words and L for record size>512 words).

4.2.1.5 Execute Audit Mode and Roll Call Mode from Cards. When the roll call mode is selected, the audit mode generates a revised roll call program file for the program unit being examined. Therefore, the roll call mode must be executed, along with the audit mode, once per program unit. If the software to be examined is on cards, the following sequence of cards is used to execute the audit mode and the roll call mode for a single program unit.

1. Standard SCOPE Job card (CM130000)
2. Standard SCOPE charge card.
3. ATTACH(TAPE4,IDFILE,ID=XXXX)
4. ATTACH(TAPE19,SYNTAXGRAPH,ID=XXXX)
5. ATTACH(AUD,AUDITBIN,ID=XXXX)

```
 6.  ATTACH(RLCL,ROLLCALLBIN,ID=XXXX)
 7.  AUD.
 8.  FTN(I=TAPE8)
 9.  LOAD(RLCL)
10.  LGO.
11.  COPYSBF(TAPE 13)
12.  COPYSBF(TAPE 14)     Used only for a value of 2 in
13.  COPYSBF(TAPE 15)     Column 1 of options card.
14.  7/8/9 End-of-record
15.  Options card
16.  Software to be examined
       .
       .
       .
17. 6/7/8/9 End-of-file
```

EXPLANATION OF CARDS

3.-4. Attach input files.
5.-6. Attach audit and roll call binary.
7. Execute audit binary.
8. Compile the revised roll call program file.
9.-10. Load the roll call binary and the revised roll call program file and execute.
11.-13. Display information written on output devices X, Y, and Z by the program unit being examined. These cards are only used when Column 1 of the options card contains a 2.
15. Options card is punched with a 2 or 3 in Column 1 and a 5 in Column 5.
16. Software must be a root program unit of a module.

The following sequence of cards (which includes cards identified in the previous sequence) is used to execute the audit mode and the roll call mode from cards for more than one program unit.

```
Cards 1-6
Cards 7-13          (for program unit 1)
REWIND,LGO.         (for program unit 2)
Cards 7-13
REWIND,LGO.         (for program unit 3)
Cards 7-13
  .
  .                 (for additional program units)
  .
7/8/9 End-of-record
Options card
Program unit 1
7/8/9
Options card
Program unit 2
```

```
7/8/9
Options card
Program unit 3
7/8/9
 .
 .          } (for additional program units)
 .
6/7/8/9 End-of-file
```

4.2.1.6 Execute Audit Mode and Roll Call Mode from Sequential File. If there is more than one program unit on a sequential tape or disk file, the audit mode can be executed only if there are end-of-record marks between program units. Since this way of storing source code is not typical, a description of the control cards needed to execute the audit and roll call mode is not provided. The user who wants to do it this way may construct the control cards by following the control card logic given for executing from cards or an update file.

4.2.1.7 Execute Audit Mode and Roll Call Mode from Update Disk or Tape File. When the roll call mode is selected, the audit mode generates a revised roll call program file for the program unit being examined. Therefore, the roll call mode must be executed, along with the audit mode, once per program unit. If the software to be examined is on an update disk file, the following sequence of cards is used to execute the audit mode and the roll call mode for a single program unit.

```
 1.  Standard SCOPE Job card (CM130000 and MT1)
 2.  Standard SCOPE charge card
 3.  ATTACH(TAPE4,IDFILE,ID=XXXX)
 4.  ATTACH(TAPE19,SYNTAXGRAPH,ID=XXXX)
 5.  ATTACH(AUD,AUDITBIN,ID=XXXX)
 6.  ATTACH(RLCL,ROLLCALLBIN,ID=XXXX)
 7.  ATTACH(OLDPL, PFN ,ID=XXXX)
 8.  UPDATE(Q,C=TAPE7)
 9.  AUD.
10.  FTN(I=TAPE8)
11.  LOAD(RLCL)
12.  LGO.
13.  COPYSBF(TAPE13)}
14.  COPYSBF(TAPE14)}      Used only for a value of 2 in
15.  COPYSBF(TAPE15)}      Column 1 of options card.
16.  7/8/9 End-of-record
17.  *C DECK
18.  7/8/9
19.  Options card
20.  6/7/8/9 End-of-file
```

EXPLANATION OF CARDS

3.-4. Attach input files.
5.-6. Attach audit and roll call binary.
7. PFN is the name of the permanent file on which the software to be examined resides.
8. Call the update utility to select the deck from the update file which the user wishes to examine.
9. Execute audit binary.
10. Compile the revised roll call program file.
11.-12. Load the roll call binary and the revised roll call program file and execute.
13.-15. Display information that has been written on output devices X, Y, and Z by the program unit being analyzed. These control cards are only used when Column 1 of the options card contains a 2.
17. DECK is the specific program unit to be examined. It must be the root program unit of a module.
19. Options card is punched with a 2 or 3 in Column 1 and a 7 in Column 5.

The following sequence of cards (which includes cards identified in the previous sequence) is used to execute the audit mode and the roll call mode from an update disk file for more than one program unit.

```
Cards 1-7
Cards 8-15          (for program unit 1)
REWIND,LGO.}        (for program unit 2)
Cards 8-15 }
REWIND,LGO.}        (for program unit 3)
Cards 8-15 }
  .        }
  .        }        (for additional program units)
  .        }
7/8/9 End-of-record
*C DECK1
7/8/9
Options card
7/8/9
*C DECK2
7/8/9
Options card
7/8/9
*C DECK3
7/8/9
Options card
  .        }
  .        }        (for additional program units)
  .        }
6/7/8/9 End-of-file
```

If the software to be examined is on an update labeled tape file, the following two cards replace card 7.

7a. VSN(TAPE7=VRNO)

7b. LABEL(OLDPL, L= NAME ,R) (VRNO/NORING)

where VRNO is the visual reel number of the tape and NAME is the tape label name. If the software to be examined is on an update stranger tape file, the following two cards replace card 7.

7a. VSN(TAPE7=VRNO)

7b. REQUEST(OLDPL,DEN,TYPE) (VRNO/NORING)

where DEN is HI, HY, or HD (556 seven track, 800 seven track, and 800 nine track, respectively) and TYPE is S or L (S for record size $\leq$512 words and L for record size >512 words).

4.2.1.8 Execute Audit Mode and Variable Precision Mode from Cards. The variable precision mode processes an entire executable program. If the executable program to be examined is on cards, the following sequence of cards is used to execute the audit mode and the variable precision mode for a single bit configuration.

1.-6. Same as 1-6 from Section 4.2.1.2
7. FTN(I=TAPE8)
8. FTN(B=VPLIB)
9. LOAD(VPLIB)
10. LGO.
11. 7/8/9 End-of-record
12. Options card
13. Executable program (without data cards)
.
.
.
14. 7/8/9
15. Variable precision function subprograms
.
.
.
16. 7/8/9
17. Data cards for executable program
.
.
.
18. 6/7/8/9 End-of-file

7. Compile the revised variable precision program file which is generated by the audit mode. This file contains references to the variable precision function subprograms.
8. Compile the variable precision function subprograms.
9.-10. Load the variable precision function subprograms and the revised variable precision program file and then execute.
12. Options card is punched with a 1 in Column 1 and a 5 in Column 5.
13. Executable program (without data cards) must contain a main program.
15. Three variable precision function subprograms (Q1REAL, Q1DPRE, and Q1COMP) for the particular bit configuration desired.
17. Data cards for executable program. If additional data is on tape or disk, the user must make such data available to the executable program.

The following sequence of cards (which includes cards identified in the previous sequence) is used to execute the audit mode and the variable precision mode from cards for more than one bit configuration.

```
Cards 1-7
Cards 8-10          (for variable precision function sub-
                     programs 1)
REWIND,VPLIB.      } (for variable precision function sub-
Cards 8-10         } programs 2)
REWIND,VPLIB.      } (for variable precision function sub-
Cards 8-10         } programs 3)
 .                 }
 .                 } (additional variable precision function
 .                 } subprograms)
7/8/9 End-of-record
Options card
Executable program (without data cards)
 .
 .
 .
7/8/9
Variable precision function subprograms 1
7/8/9
Data cards for executable program
7/8/9
Variable precision function subprograms 2
7/8/9
Data cards for executable program
7/8/9
Variable precision function subprograms 3
```

```
7/8/9
Data cards for executable program
  .
  .              } (additional variable precision function
  .              } subprograms)
6/7/8/9 End-of-file
```

4.2.1.9 Execute Audit Mode and Variable Precision Mode from Sequential Disk or Tape File. The variable precision mode processes an entire executable program. If the executable program to be examined is on a sequential disk file and data is on cards, the following sequence of cards is used to execute the audit mode and the variable precision mode for a single bit configuration.

```
1.-5.  Same as 1-5 from Section 4.2.1.8
6.     ATTACH(TAPE7, PFN ,ID=XXXX)
7.     AUD.
8.     FTN(I=TAPE8)
9.     FTN(B=VPLIB)
10.    LOAD(VPLIB)
11.    LGO.
12.    7/8/9 End-of-record
13.    Options card
14.    7/8/9
15.    Variable precision function subprograms
         .
         .
         .
16.    7/8/9
17.    Data cards for executable program
         .
         .
         .
18.    6/7/8/9 End-of-file
```

EXPLANATION OF CARDS

6. PFN is the name of the permanent file on which the executable program resides.
7. Execute the audit binary.
8. Compile the revised variable precision program file which is generated by the audit mode. This file contains references to the variable precision function subprograms.
9. Compile the variable precision function subprograms.
10.-11. Load the variable precision function subprograms and the revised variable precision program file and then execute.
13. Options card is punched with a 1 in Column 1 and a 7 in Column 5.

15. Three variable precision function subprograms (Q1REAL, Q1DPRE, and Q1COMP) for the particular bit configuration desired.

The following sequence of cards (which includes cards identified in the previous sequence) is used to execute the audit mode and the variable precision mode from a sequential disk file (executable program on file and data on cards) for more than one bit configuration.

```
Cards 1-8
Cards 9-11          (for variable precision function sub-
                    programs 1)
REWIND,VPLIB.       (for variable precision function sub-
Cards 9-11          programs 2)
REWIND, VPLIB       (for variable precision function sub-
Cards 9-11          programs 3)
 .
 .                  (for additional variable precision
 .                  function subprograms)
7/8/9 End-of-record
Options card
7/8/9
Variable precision function subprograms 1
7/8/9
Data cards for executable program
7/8/9
Variable precision function subprograms 2
7/8/9
Data cards for executable program
7/8/9
Variable precision function subprograms 3
7/8/9
Data cards for executable program
 .
 .                  (for additional variable precision
 .                  function subprograms)
6/7/8/9 End-of-file
```

If the executable program is on a labeled or a stranger sequential tape file, substitute two cards for Card 6, as described in Section 4.2.1.3.

4.2.1.10 Execute Audit Mode and Variable Precision Mode from Update Disk or Tape File. The variable precision mode processes an entire executable program. If the executable program is on an update disk file and data is on cards, the following sequence of cards is used to execute the audit mode and the variable precision mode for a single bit configuration.

1.-5. Same as 1-5 from Section 4.2.1.8.
6. ATTACH(OLDPL, PFN ,ID=XXXX)

```
 7.     UPDATE(Q,C=TAPE7)
 8.     AUD.
 9.     FTN(I=TAPE8)
10.     FTN(B=VPLIB)
11.     LOAD(VPLIB)
12.     LGO.
13.     7/8/9 End-of-record
14.     *C DECK1,DECK2,.....,DECKn
15.     7/8/9
16.     Options Card
17.     7/8/9
18.     Variable precision function subprograms
          .
          .
          .
19.     7/8/9
20.     Data cards for executable program
          .
          .
          .
21.     6/7/8/9 End-of-file
```

EXPLANATION OF CARDS

6. PFN is the name of the permanent file on which the executable program resides.

7. Call the update utility to select the decks from the update file which make up the executable program. If the executable program consists of all decks on the update file, replace the Q with an F.

8. Execute the audit binary.

9. Compile the revised variable precision program file which is generated by the audit mode. This file contains references to the variable precision function subprograms.

10. Compile the variable precision function subprograms.

11.-12. Load the variable precision function subprograms and the revised variable precision program file and execute.

14. DECK1,DECK2,.....,DECKn are the specific decks that make up the executable program. One and only one deck must be a main program. If an F is on the update card (card 7), omit this card.

16. Options card is punched with a 1 in Column 1 and a 7 in Column 5.

18. Three variable precision function subprograms (Q1REAL, Q1DPRE, and Q1COMP) for the particular bit configuration desired.

20. Data cards for the executable program. If additional data is on disk or tape, the user must make such data available to the executable program.

The following sequence of cards (which includes cards identified in the previous sequence) is used to execute the audit mode and the variable precision mode from an update disk file (executable program on file and data on cards) for more than one bit configuration.

```
Cards 1-9
Cards 10-12      (for variable precision function sub-
                  programs 1)
REWIND,VPLIB    } (for variable precision function sub-
Cards 10-12     } programs 2)
REWIND,VPLIB    } (for variable precision function sub-
Cards 10-12     } programs 3)
  .             }
  .             } (for additional variable precision
  .             } function subprograms)
7/8/9 End-of-record
Cards 14-16
7/8/9
Variable precision function subprograms 1
7/8/9
Data cards for executable program
7/8/9
Variable precision function subprograms 2
7/8/9
Data cards for executable program
7/8/9
Variable precision function subprograms 3
7/8/9
Data cards for executable program
  .             }
  .             } (for additional variable precision
  .             } function subprograms)
6/7/8/9 End-of-file
```

If the executable program is on a labeled or a stranger update tape file, substitute two cards for Card 6, as described in Section 4.2.1.4.

4.2.2 UNIVAC 1108. The control cards listed in this section are applicable to the UNIVAC 1100 series with EXEC-8 operating system.

4.2.2.1 Initial File Creation. The following four files must be created initially:

1. Interface Definition file
2. Syntax graph file
3. Absolute binary version of the audit mode source software.
4. Relocatable binary version of the roll call mode source software.

To create the Interface Definition file, execute program SESLIST and catalogue the output file with a file name of IDFILE. To create the syntax graph file, execute program GRAPH and catalogue the output file with a file name of SYNTAXGRAPH. The audit binary version (with a file name of AUDITBIN) and the roll call binary version (with a file name of ROLLCALLBIN) are created from the source unlabeled tape (with a file name of TAPE) using the following control cards:

```
 1.  Standard EXEC-8 Job card
 2.  @ASG,TJ TAPE.,8C,AUDIT
 3.  @ASG,UP AUDITBIN.
 4.  @ASG,T COMPILE.
 5.  @ASG,T AUDITPL.
 6.  @ASG,T RLCL.
 7.  @ASG,UP ROLLCALLBIN.
 8.  @ASG,T AUDIT.
 9.  @COPY,G TAPE.,COMPILE.
10.  @COPY,G TAPE.,AUDITPL.
11.  @COPY,G TAPE.,RLCL.
12.  @ADD COMPILE.
13.  @MAP,SIX ,AUDITBIN.AUD
14.  IN AUDIT.
15.  @FOR RLCL.ROLCHK,ROLLCALLBIN.ROLCHK
16.  @FOR RLCL.CMPARE,ROLLCALLBIN.CMPARE
17.  @FOR RLCL.MODID,ROLLCALLBIN.MODID
18.  @FIN
```

EXPLANATION OF CARDS

```
 2.       Mount AUDIT source tape.
 3.       Assign a catalogued file for the audit mode absolute
          binary.
 4.       Assign a temporary file containing @FOR cards to
          compile the AUDIT source.
 5.       Assign a temporary file for the audit mode source.
 6.       Assign a temporary file for the roll call mode source.
 7.       Assign a catalogued file for the roll call binary.
 8.       Assign a temporary file for the audit binary.
 9.       Copy the @FOR cards from tape.
10.       Copy the audit mode source from tape.
11.       Copy the roll call source from tape.
12.       Compile the audit source.
13.-14.   Call the MAP processor to create an absolute binary
          audit mode file with a name of AUDITBIN.
15.-17.   Compile the roll call mode source and create a
          binary roll call mode file with a name of ROLLCALLBIN.
```

4.2.2.2 Execute Audit Mode from Cards. If the software to be examined is on cards, the following sequence of cards is used to execute the audit mode.

```
 1.  Standard EXEC-8 Job card.
 2.  @ASG,A IDFILE.
 3.  @ASG,A SYNTAXGRAPH.
 4.  @ASG,A AUDITBIN.
 5.  @USE 4.,IDFILE.
 6.  @USE 19.,SYNTAXGRAPH.
 7.  @XQT AUDITBIN.AUD
 8.  Options card
 9.  Software to be examined
       .
       .
       .
10.  @FIN
```

EXPLANATION OF CARDS

2.-3. Assign input files.
4. Assign audit binary.
5.-6. Assign logical units to input files.
7. Execute the audit binary.
8. Options card is punched with a 1 in Column 1 and a 5 in Column 5.
9. At least one program unit is needed. Only one main program is permitted.

4.2.2.3 Execute Audit Mode from Disk or Tape File. The file to be examined must contain at least one program unit. Only one main program is permitted. If the software to be examined is on a disk or a tape SDF formatted file (source), the following sequence of cards is used to execute the audit mode.

```
 1.  Standard EXEC-8 Job card.
 2.  @ASG,A IDFILE.
 3.  @ASG,A SYNTAXGRAPH.
 4.  @ASG,A AUDITBIN.
 5.  Assign disk or tape with a file name of AUDFILE.
 6.  @USE 4.,IDFILE.
 7.  @USE 19.,SYNTAXGRAPH.
 8.  @USE 7.,AUDFILE.
 9.  @XQT AUDITBIN.AUD
10.  Options card
11.  @FIN
```

EXPLANATION OF CARDS

5.,8. AUDFILE is the file name on which the software to be examined resides.
10. Options card is punched with a 1 in Column 1 and a 7 in Column 5.

(See Section 4.2.2.2 for an explanation of the other cards.)

If the software to be examined is on a disk or tape program file (elements), the following sequence of cards (which includes cards identified in the previous sequence) is used to execute the audit mode.

```
Cards 1-5
@ASG,T 7.
Cards 6-7
@DATA,I 7.
@ADD,D AUDFILE.ELT1
@ADD,D AUDFILE.ELT2
        .
        .
        .
@END
Cards 9-11
```

EXPLANATION OF CARDS

An @ADD,D card is needed for each element to be examined. ELT1, ELT2, etc., are the element names.

4.2.2.4 Execute Audit Mode and Roll Call Mode from Cards. When the roll call mode is selected, the audit mode generates a revised roll call program file for the program unit being examined. Therefore, the roll call mode must be executed, along with the audit mode, once per program unit. If the software to be examined is on cards, the following sequence of cards is used to execute the audit mode and the roll call mode for a single program unit.

```
 1.  Standard EXEC-8 Job card.
 2.  @ASG,A IDFILE.
 3.  @ASG,A SYNTAXGRAPH.
 4.  @ASG,A AUDITBIN.
 5.  @ASG,A ROLLCALLBIN.
 6.  @USE 4.,IDFILE.
 7.  @USE 19.,SYNTAXGRAPH.
 8.  @XQT AUDITBIN.AUD
 9.  Options Card
10.  Software to be examined
                .
                .
                .
11.  @END
12.  @ADD 8.
13.  @MAP,IX ROLL
14.  IN TPF$.
15.  IN ROLLCALLBIN.
16.  @XQT ROLL
```

```
17.  @DATA,L  13.
18.  @END
19.  @DATA,L  14.   Used only for a value of 2 in Column 1
20.  @END           of options card.
21.  @DATA,L  15.
22.  @END
23.  @FIN
```

EXPLANATION OF CARDS

2.-3.	Assign input files.
4.-5.	Assign audit and roll call binary.
6.-7.	Assign logical units to input files.
8.	Execute the audit binary.
9.	Options card is punched with a 2 or 3 in Column 1 and a 5 in Column 5.
10.	Software must be a root program unit of a module.
12.	Compile the revised roll call program file.
13.-15.	Call the MAP processor to create an absolute binary roll call mode file composed of the file in 13. and the binary roll call software.
16.	Execute the absolute binary roll call mode file.
17.-22.	Display information written on output devices X, Y, and Z by the program unit being examined. These cards are only used when Column 1 of the options card contains a 2.

The following sequence of cards (which includes cards identified in the previous sequence) is used to execute the audit mode and the roll call mode from cards for more than one program unit.

```
Cards 1-7
Cards 8-9
Program unit 1   (for program unit 1)
Cards 11-22
Cards 8-9
Program unit 2   (for program unit 2)
Cards 11-22
 .
 .               (for additional program units)
 .
@FIN
```

4.2.2.5 Execute Audit Mode and Roll Call Mode from Disk or Tape File. When the roll call mode is selected, the audit mode generates a revised roll call program file for the program unit being examined. Therefore, the roll call mode must be executed, along with the audit mode, once per program unit. If the software to be examined is on a disk or tape program file (elements), the following sequence of cards is used to execute the audit mode and the roll call mode for a single program unit.

1.-5. Same as 1-5 of Section 4.2.2.4
6. Assign disk or tape with a file name of ROLLFILE.
7. @ASG,T 7.
8.-9. Same as 6-7 of Section 4.2.2.4
10. @DATA,I 7.
11. @ADD,D ROLLFILE.ELTNAME
12. @END
13.-14. Same as 8-9 of Section 4.2.2.4 (Options card has a 7 in Column 5)
15.-27. Same as 11-23 of Section 4.2.2.4.

EXPLANATION OF CARDS

6.,11. ROLLFILE is the file name on which the software to be examined resides. ELTNAME is the element name of the element.

The previous sequence of cards is used if only one program unit is to be examined. The following sequence of cards (which includes cards identified in the previous sequence) is used to execute the audit mode and the roll call mode from a disk or tape program file (elements) for more than one program unit.

```
Cards 1-9
Cards 10-12 }  (Card 11 with the first element name)
Cards 13-26 }
Cards 10-12 }  (Card 11 with the second element name)
Cards 13-26 }
  .         }
  .         }  (for additional program units)
  .         }
@FIN
```

4.2.2.6 Execute Audit Mode and Variable Precision Mode from Cards. The variable precision mode processes an entire executable program. If the executable program to be examined is on cards, the following sequence of cards is used to execute the audit mode and the variable precision mode for a single bit configuration.

1. Standard EXEC-8 Job card.
2. @ASG,A IDFILE.
3. @ASG,A SYNTAXGRAPH.
4. @ASG,A AUDITBIN.
5. @USE 4.,IDFILE.
6. @USE 19.,SYNTAXGRAPH.
7. @XQT AUDITBIN.AUD
8. Options card
9. Executable program (without data cards)

.
.
.

```
10.  @FOR,IS Q1REAL
11.  Function subprogram Q1REAL
                  .
                  .
                  .
12.  @FOR,IS Q1COMP
13.  Function subprogram Q1COMP
                  .
                  .
                  .
14.  @FOR,IS Q1DPRE
15.  Function subprogram Q1DPRE
                  .
                  .
                  .
16.  @ADD 8.
17.  @MAP,IX VARPRC
18.  IN TPF$.
19.  @XQT VARPRC
20.  Data cards for executable program
                  .
                  .
                  .
21.  @FIN
```

EXPLANATION OF CARDS

2.-3.	Assign input files.
4.	Assign audit binary.
5.-6.	Assign I/O devices to input files.
7.	Execute the audit binary.
8.	Options card is punched with a 1 in Column 1 and a 5 in Column 5.
9.	Executable program (without data cards) must contain a main program.
10.-15.	Compile variable precision function subprograms Q1REAL, Q1COMP, and Q1DPRE. These subprograms must all be for the same bit configuration.
16.	Compile the revised variable precision program file which contains references to the variable precision function subprograms.
17.-18.	Call the MAP processor to create the absolute binary variable precision file.
19.	Execute the variable precision mode.
20.	Data cards for executable program. If additional data is on tape or disk, the user must make this data available to the executable program.

The following sequence of cards (which includes cards identified in the previous sequence) is used to execute the audit mode and the variable precision mode from cards for more than one bit configuration.

```
Cards 1-8
Executable program (without data cards)
10-15 (for bit configuration 1)
Cards 16-20
10-15 (for bit configuration 2)
Cards 16-20
.
.      (for additional bit configurations)
.
@FIN
```

EXPLANATION OF CARDS

Each group of variable precision function subprograms must be for a different bit configuration.

4.2.2.7 Execute Audit Mode and Variable Precision Mode from Disk or Tape File. The variable precision mode processes an entire executable program. If the executable program is on a disk or tape SDF formatted file (source) and the data is on cards, the following sequence of cards is used to execute the audit mode and the variable precision mode for a single bit configuration.

1.-10. Same as 1-10 of Section 4.2.2.3. (Cards 5 and 8 have a file name of VPFILE.)
11.-22. Same as 10-21 of Section 4.2.2.6

EXPLANATION OF CARDS

5.,8. VPFILE is the name of the file on which the executable program resides.

The following sequence of cards (which includes cards identified in the previous sequence) is used to execute the audit mode and the variable precision mode from tape or disk for more than one bit configuration.

```
Cards 1-10
11-16 (for bit configuration 1)
Cards 17-21
11-16 (for bit configuration 2)
Cards 17-21
.
.  (for additional bit configurations)
.
@FIN
```

EXPLANATION OF CARDS

Each group of variable precision function subprograms must be for a different bit configuration.

If the executable program to be examined is on a disk or tape program file (elements), the following sequence of cards is used to execute the audit mode and the variable precision mode for a single bit configuration.

```
1.-4.     Same as 1-4 of Section 4.2.2.3
5.        Assign disk or tape with a file name of
          VPFILE.
6.        @ASG,T 7.
7.-8.     Same as 6-7 of Section 4.2.2.3
9.        @DATA,I 7.
10.       @ADD,D VPFILE.ELT1
          @ADD,D VPFILE.ELT2
                  .
                  .
                  .
11.       @END
12.-13.   Same as 9-10 of Section 4.2.2.3
14.-25.   Same as 10-21 of Section 4.2.2.6
```

EXPLANATION OF CARDS

An @ADD,D card is needed for each element that makes up the executable program. ELT1,ELT2, etc., are the element names. VPFILE is the name of the file on which the elements reside.

The following sequence of cards (which includes cards identified in the previous sequence) is used to execute the audit mode and the variable precision mode from tape or disk for more than one bit configuration.

```
1-13
14-19  (for bit configuration 1)
Cards 20-24
14-19  (for bit configuration 2)
Cards 20-24
.
.      (for additional bit configurations)
.
@FIN
```

EXPLANATION OF CARDS

Each group of variable precision function subprograms must be for a different bit configuration.

4.2.3 IBM 360. The control cards listed in this section are applicable to the OS/360 operating system for the IBM 360. Changes should be made as appropriate for other operating systems.

4.2.3.1 Initial Data Set Creation. The following four permanent data sets must be created initially:

1. Interface Definition file
2. Syntax graph file
3. Binary version of the audit mode source software
4. Binary version of the roll call mode source software

To create the Interface Definition file, execute program SESLIST and catalogue the output file with a permanent data set name of IDFILE. To create the syntax graph file, execute program GRAPH and catalogue the output file with a permanent data set name of SYNTXGRF. The audit binary version (with a permanent data set name of AUDITBIN) and the roll call binary version (with a permanent data set name of RLCLBIN) are created from a labeled tape using the following control cards.

```
1.  Standard IBM Job card
2.  // EXEC PGM=CSDS,PARM='LRECL=3120'
3.  //SYSPRINT DD SYSOUT=A
4.  //INPUT DD DSN=NSR.SOURCE.AMK.TAPE,VOL=SER=XXXX,
    UNIT=TAPE9,
5.  // LABEL=(1,SL),DCB=(RECFM=U,BLKSIZE=3120),DISP=
    (OLD,KEEP)
6.  //OUTPUT DD DSN=&AUDITBIN,DISP=(NEW,PASS),UNIT=
    TEMP,
7.  // DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120),SPACE=
    (3120,(200,5))
8.  // EXEC PGM=CSDS,PARM='LRECL=4000'
9.  //SYSPRINT DD SYSOUT=A
10. //INPUT DD DSN=NSR.SOURCE.AMK.TAPE,VOL=SER=XXXX,
    UNIT=TAPE9,
11. // LABEL=(2,SL),DCB=(RECFM=U,BLKSIZE=4000),DISP=
    (OLD,KEEP)
12. //OUTPUT DD DSN=&RLCLBIN,DISP=(NEW,PASS),UNIT=
    TEMP,
13. // DCB=(RECFM=FB,LRECL=80,BLKSIZE=4000),SPACE=
    (4000,(5,5))
14. // EXEC FHL,LMOD='Q1.Q2.AUDITBIN(AUD)',DSP=NEW,
    DEV=SAVE,
15. // TIME=(5,0),REGION.F=250K,PARM.L=MAP,REGION.L=
    200K
16. //F.SYSPRINT DD OUTLIM=20000
17. //F.SYSIN DD DSN=&AUDITBIN,DISP=(OLD,DELETE)
18. //L.SYSLMOD DD SPACE=(3156,(600,50,15))
19. // EXEC FHL,PARM.L=MAP,LMOD='Q1.Q2.RLCLBIN(ROLL)',
    DSP=NEW,DEV=SAVE
20. //F.SYSIN DD DSN=&RLCLBIN,DISP=(OLD,DELETE)
21. //L.SYSLMOD DD SPACE=(100,(10,10,10))
22. /*
```

EXPLANATION OF CARDS

2. Call the utility CSDS (copy sequential data set) to copy the audit mode source from tape.
4.-5. Describe the input data set, which is the source tape. XXXX is the volume serial number of the tape.
6.-7. Describe the output data set, which will be a temporary data set.
8. Call the utility CSDS to copy the roll call mode source from tape.
10.-13. Same as 4-7.
14.-15. Call the FHL catalogued procedure (H compile and link edit) to create the audit load module which is to be catalogued.
17. The audit source is input to the compiler.
18. Allocate space for the audit load module.
19. Call the FHL catalogued procedure to create and catalog the roll call load module.
20. The roll call source is input to the compiler.
21. Allocate space for the roll call load module.

4.2.3.2 Execute Audit Mode from Cards. If the software to be examined is on cards, the following sequence of cards is used to execute the audit mode.

```
 1. Standard IBM Job card.
 2. // EXEC PGM=AUD,REGION=250K
 3. //STEPLIB DD DSN=Q1.Q2.AUDITBIN,DISP=SHR
 4. //FT06F001 DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,
    BLKSIZE=3458)
 5. //FT04F001 DD DSN=Q1.Q2.IDFILE,DISP=SHR
 6. //FT19F001 DD DSN=Q1.Q2.SYNTXGRF,DISP=SHR
 7. //FT08F001 DD DSN=&OUTPT,DISP=(NEW,DELETE),UNIT=TEMP,
 8. // DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120),SPACE=(3120,
    (10,10))
 9. //FT05F001 DD *
10. Options Card
11. Software to be examined
                .
                .
                .
12. /*
```

EXPLANATION OF CARDS

2.-3. Get audit binary.
4. Assign the print file.
5. Assign the Interface Definition file to logical unit 4.
6. Assign the syntax graph file to logical unit 19.
7.-8. Assign temporary file for audit revised program file.

10. Options card is punched with a 1 in Column 1 and a 5 in Column 5.
11. At least one program unit is needed. Only one main program is permitted.

4.2.3.3 Execute Audit Mode from a Sequential Tape or Disk Data Set. A sequential data set must contain at least one program unit. If there is more than one program unit, all program units must be consecutively stored. Only one main program is permitted. If the software to be examined is on a sequential tape or disk data set, the following sequence of cards is used to execute the audit mode.

```
1.-8.  Same as 1-8 of Section 4.2.3.2
9.     Assign tape or disk sequential data set to logical
       unit 2.
10.    //FT05F001 DD *
11.    Options card
12.    /*
```

EXPLANATION OF CARDS

11. Options card is punched with a 1 in Column 1 and a 2 in Column 5.

4.2.3.4 Execute Audit Mode from a Partitioned Disk Data Set. The partitioned data set must contain at least one program unit. Only one main program is permitted per execution. If the software to be examined is on a partitioned disk data set, the following sequence of cards is used to execute the audit mode.

```
1.      Standard IBM Job card.
2.      // EXEC LISTPCH,LIB='Q1.Q2.NAME'
3.      //SYSPRINT DD DUMMY,DCB=(RECFM=VBA,LRECL=137,
        BLKSIZE=3429)
4.      //SYSPUNCH DD DSN=&DATA,DISP=(NEW,PASS),UNIT=TEMP,
5.      // DCB=(RECFM=FB,LRECL=80,BLKSIZE=3200),SPACE=
        (3200,(50,10))
6.      //SYSIN DD *
7.      MEMBER1
        MEMBER2
          .
          .
          .
        MEMBERn
8.-14.  Same as 2-8 of Section 4.2.3.2
15.     //FT02F001 DD DSN=&DATA,DISP=(OLD,DELETE)
16.     //FT05F001 DD *
17.     Options card
18.     /*
```

EXPLANATION OF CARDS

2. Call the catalogued procedure LISTPCH to select the desired members from the partitioned data set. 'Q1. Q2.NAME' is the data set name. This procedure uses the standard IBM utility IEBCOPY.
3. Assigns the print file data set.
4.-5. Data set on which the members will reside.
7. Data set members to be examined, punched one name to a card anywhere on the card.
15. Logical unit 2 data set on which the members reside.

4.2.3.5 Execute Audit Mode and Roll Call Mode from Cards. When the roll call mode is selected, the audit mode generates a revised roll call program file for the program unit being examined. Therefore, the roll call mode must be executed, along with the audit mode, once per program unit. If the software to be examined is on cards, the following sequence of cards is used to execute the audit mode and the roll call mode for a single program unit.

```
1.  Standard IBM Job card
2.  // EXEC PGM=AUD,REGION=250K
3.  //STEPLIB DD DSN=Q1.Q2.AUDITBIN,DISP=SHR
4.  //FT06F001 DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=133,
    BLKSIZE=3458)
5.  //FT04F001 DD DSN=Q1.Q2.IDFILE,DISP=SHR
6.  //FT19F001 DD DSN=Q1.Q2.SYNTXGRF,DISP=SHR
7.  //FT08F001 DD DSN=&OUTPT,DISP=(NEW,PASS),UNIT=TEMP,
8.  // DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120),SPACE=
    (3120,(10,10))
9.  //FT09F001 DD DSN=&LIST1,DISP=(NEW,PASS),UNIT=TEMP,
10. // DCB=(RECFM=VBS,BLKSIZE=100),SPACE=(100, (10,10))
11. //FT05F001 DD *
12. Options Card
13. Program Unit
          .
          .
          .
14. // EXEC FHLG,PARM.L=MAP
15. //F.SYSIN DD DSN=&OUTPT,DISP=(MOD,DELETE)
16. //L.DD1 DD DSN=Q1.Q2.RLCLBIN,DISP=SHR
17. //L.SYSIN DD *
18.       INCLUDE DD1(ROLL)
19.        ENTRY MAIN
20. //G.FT09F001 DD DSN=&LIST1,DISP=(OLD,DELETE)
21. //G.FT03F001 DD DSN=&LIST2,DISP=(NEW,PASS), UNIT=TEMP,
22. // DCB=(RECFM=VBS,BLKSIZE=100),SPACE=(100,(10,10))
```

```
23.  //G.FT03F002 DD DSN=&LIST3,DISP=(NEW,PASS),UNIT=TEMP,
24.  Same as 22
25.  //G.FT03F003 DD DSN=&LIST4,DISP=(NEW,PASS),UNIT=TEMP,
26.  Same as 22
     .
     .  (for temporary data sets 4 through 12)
     .
45.  //G.FT03F013 DD DSN=&LIST14,DISP=(NEW,PASS),UNIT=TEMP,
46.  Same as 22
47.  //G.FT10F001 DD DSN=&X1,DISP=(NEW,PASS),UNIT=TEMP,
48.  // DCB=(RECFM=FB,LRECL=80,BLKSIZE=3120),SPACE=(3120,
     (10,10))
49.  //G.FT11F001 DD DSN=&Y1,DISP=(NEW,PASS),UNIT=TEMP,
50.  Same as 48
51.  //G.FT12F001 DD DSN=&Z1,DISP=(NEW,PASS),UNIT=TEMP,
52.  Same as 48
53.  //G.FT13F001 DD DSN=&X2,DISP=(NEW,PASS),UNIT=TEMP,
54.  Same as 48
55.  //G.FT14F001 DD DSN=&Y2,DISP=(NEW,PASS),UNIT=TEMP,
56.  Same as 48
57.  //G.FT15F001 DD DSN=&Z2,DISP=(NEW,PASS),UNIT=TEMP,
58.  Same as 48
59.  // EXEC PGM=IEBPTPCH
60.  //SYSPRINT DD SYSOUT=A
61.  //SYSUT1 DD DSN=&X2,DISP=(OLD,DELETE)
62.  //        DD DSN=&Y2,DISP=(OLD,DELETE)
63.  //        DD DSN=&Z2,DISP=(OLD,DELETE)
64.  //SYSUT2 DD SYSOUT=A,DCB=(RECFM=FBA,LRECL=121,
     BLKSIZE=3509)
65.  //SYSIN DD *
66.        PRINT MAXFLDS=1
67.        RECORD FIELD=(80,,,5)
68.  /*
```

Note: Cards 47-67 are used only for a value of 2 in column 1 of the options card.

EXPLANATION OF CARDS

1.-8. Same as 1-8 of Section 4.2.3.2.

9.-10. Assign a data set for the audit module list (Section 3.2.2.2).

12. Options card is punched with a 2 or 3 in Column 1 and a 5 in Column 5.

13. Program unit must be a root program unit of a module.

14. Call the FHLG catalogued procedure to compile, link edit, and execute the revised roll call program file.

15. Declare the data set &OUTPT as input to the FORTRAN compiler. This data set is generated by the roll call

source and contains the main program ROLCAL and the revised program unit (containing calls to ROLCHK).

16. Declare the roll call source as input to the link editor.

17.-19. Link editor input cards.

20. Declare the audit module list data set.

21.-52. Declare temporary data sets. (Cards 47-52 used only for a value of 2 in column 1 of the options card).

53.-58. Declare roll call output data sets which are the contents of SESCOMP I/O units X, Y, and Z. (Used only for value of 2 in column 1 of options card).

59.-67. Display the contents of I/O units X, Y, and Z (Used only for value of 2 in column 1 of options card).

In order to execute the audit mode and the roll call mode from cards for more than one program unit, it is best to run each program unit as a separate job since it is more complicated to execute several program units in a single job.

4.2.3.6 Execute Audit Mode and Roll Call Mode from a Partitioned Disk Data Set. When the roll call mode is selected, the audit mode generates a revised roll call program file for the program unit being examined. Therefore, the roll call mode must be executed, along with the audit mode, once per program unit. If the software to be examined is on a partitioned disk data set, the following sequence of cards is used to execute the audit mode and the roll call mode for a single program unit.

```
1.-6.    Same as 1-6 of Section 4.2.3.4
7.       MEMBER1
8.-16.   Same as 2-10 of Section 4.2.3.5
17.      //FT02F001 DD DSN=&DATA,DISP=(OLD,DELETE)
18.      //FT05F001 DD *
19.      Options Card
20.-74.  Same as 14-68 of Section 4.2.3.5.
```

In order to execute the audit mode and the roll call mode from a partitioned data set for more than one program unit, it is best to run each program unit as a separate job since it is more complicated to execute several program units in a single job.

4.2.3.7 Execute Audit Mode and Variable Precision Mode from Cards. The variable precision mode processes an entire executable program. If the executable program to be examined is on cards, the following sequence of cards is used to execute the audit mode and the variable precision mode for a single bit configuration.

1.-10. Same as 1-10 of Section 4.2.3.2 (but change DELETE to PASS on card 7)

```
11.     Executable program (without data cards)
                          .
                          .
                          .
12.     // EXEC FHG
13.     //F.SYSIN DD DSN=&OUTPT,DISP=(MOD,DELETE)
14.     // DD *
15.     Variable precision function subprograms
                          .
                          .
                          .
16.     //G.SYSIN DD *
17.     Data cards for executable program
                          .
                          .
                          .
18.     /*
```

EXPLANATION OF CARDS

11. Executable program (without data cards) must contain a main program.

12. Call the FHG catalogued procedure to compile and execute the executable program.

13. Compile the revised variable precision program file which is generated by the audit mode. This file contains references to the variable precision function subprograms.

14.-15. Compile the variable precision function subprograms (Q1REAL, Q1DPRE, and Q1COMP) for the particular bit configuration desired.

17. Data cards for executable program. If additional data is on tape or disk, the user must make such data available to the executable program.

The following sequence of cards (which includes cards identified in the previous sequence) is used to execute the audit mode and the variable precision mode from cards for more than one bit configuration.

```
Cards 1-10
Executable program (without data cards)
Cards 12-14
Variable precision function subprograms 1
16-17
Cards 12-14
Variable precision function subprograms 2
16-17
  .
  .  (for additional bit configurations)
  .
/*
```

4.2.3.8 Execute Audit Mode and Variable Precision Mode from a Sequential Tape or Disk Data Set. The variable precision mode processes an entire executable program. If the executable program to be examined is on a sequential tape or disk data set and data is on cards, the following sequence of cards is used to execute the audit mode and the variable precision mode for a single bit configuration.

1.-11. Same as 1-11 of Section 4.2.3.3 (but change DELETE to PASS on card 7).
12.-18. Same as 12-18 of Section 4.2.3.7.

The following sequence of cards (which includes cards identified in the previous sequence) is used to execute the audit mode and the variable precision mode from a sequential tape or disk data set (executable program on file and data on cards) for more than one bit configuration.

```
Cards 1-11
Cards 12-14
Variable precision function subprograms 1
16-17
Cards 12-14
Variable precision function subprograms 2
16-17
.
.  (for additional bit configurations)
.
/*
```

4.2.3.9 Execute Audit Mode and Variable Precision Mode from a Partitioned Disk Data Set. The variable precision mode processes an entire executable program. If the executable program is on a partitioned disk data set and data is on cards, the following sequence of cards is used to execute the audit mode and the variable precision mode for a single bit configuration.

1.-17. Same as 1-17 of Section 4.2.3.4 (but change DELETE to PASS on card 13)
18.-24. Same as 12-18 of Section 4.2.3.7.

EXPLANATION OF CARDS

7. The data set members must together make up an executable program. One member must be a main program.

The following sequence of cards (which includes cards identified in the previous sequence) is used to execute the audit mode and the variable precision mode from a partitioned disk data set (executable program on file and data on cards) for more than one bit configuration.

```
Cards 1-17
Cards 18-20
Variable precision function subprograms 1
22-23
Cards 18-20
Variable precision function subprograms 2
22-23
.
.  (for additional bit configurations)
.
/*
```

4.3 Verification. For each program unit examined, AUDIT prints out any deviations from the SESCOMP standards. Most of these deviations must be corrected if the program unit is to conform with the SESCOMP standards. However, the flow analysis issues diagnostics that must be verified. A variable is indicated as being referenced but not defined along some path. The flow analysis merely indicates a possible undefined variable. To determine whether or not the particular path can be taken and whether or not the variable along the path is actually referenced and undefined, this path should be analyzed by the user.

4.4 Special Maintenance Programs.

4.4.1 Program SESLIST. The Interface Definition file (Section 3.2.1.3) is created by program SESLIST, which reads card input and creates a tape or disk file. SESLIST reads the input cards, packs the card images, and writes them out unformatted onto logical unit 4. The file should be saved for later use by AUDIT. Each time the Interface Definition file is to be changed, SESLIST must be executed. Section 3.2.1.3 contains a description of the Interface Definition file and the card input for SESLIST.

4.4.2 Program GRAPH. The syntax graph file (Section 3.2.1.4) is created by program GRAPH, which reads card input and creates a tape or disk file. GRAPH reads the input cards and writes the syntax graph as an unformatted file onto logical unit 4. The user need not concern himself with the card data used to construct the file. Since the data remains constant, the user need only create the file once, by executing GRAPH. Sections 2.1.5 and 3.2.1.4 contain a description of the syntax graph.

4.5 Error Conditions. The designers of the AUDIT software have assigned sizes to many of the arrays so that core storage will not be overburdened. Reasonable limits were chosen so that most software could be examined without any array overflow. It is possible, however, that some software may cause an array overflow. In this case the array size would have to be changed so that the software could be processed properly. When an array overflows, a message is printed indicating the type of the array. In most cases AUDIT processing then terminates.

If the DO stack array overflows, the DO loop processing terminates but the rest of the processing continues. If an array overflows during the flow analysis, the flow analysis terminates but the rest of the processing continues. Following are the arrays to be watched for possible overflow during processing.

1. Basic block table - IBLOCK(2500) in COMMON block BASBLK.
2. Symbol table - IDTBL(8,500) or IDTBL(11,500) in blank COMMON.
3. DO stack - ISTACK(4,50) in COMMON block DOLOOP.
4. Branch list - ISTCK(100) in SUBROUTINE FLOWCK.
5. Block list - FLWLST(100) in SUBROUTINE FLOWCK.
6. GIRS memory - NODSPC(1000) in COMMON block LVVTR1
   LSTSPC(1000) in COMMON block LVVTR2
   LNKSPC(1000) in COMMON block LVVTR3
   FLGSPC(1000) in COMMON block LVVTR4
7. Subroutine table - ISUBS(100) in COMMON block GLOBAL
8. External reference table - EXTTBL(100) in COMMON block GLOBAL.
9. COMMON block table - BLKTBL(200) in COMMON block GLOBAL.
10. Statement number table - STATRA(2,200) in COMMON block LABELS.
11. Statement reference table - IFNCRA(5,12) in COMMON block FUNC.
12. Statement reference location table - FNCLOC(5) in COMMON block FUNC.
13. Expression variables table - IARGS(50) in COMMON block FUNC.
14. Statement function table - ISTFNC(10) in COMMON block STFUNC.
15. Interface Definition file tables - ISUBLT(2,200), ISUBLT(3,200), or ISUBLT(4,200) and INTFAC(300), INTFAC(500), or INTFAC(600) in COMMON block LIST.
16. Encoded expression table - STR(500) in COMMON block STRING.
17. Equivalence table - IQUIV(100) in SUBROUTINE CHKLST.

The subroutine, external reference, and COMMON block tables are global tables which store information on all the program units being examined. All other arrays store information for a single program unit and are reused for each new program unit being examined.

REFERENCES

1. "SESCOMPSPEC1 SESCOMP: Introduction and Glossary; Executable Program and Main Program; February 1975;" Navy Surface Effect Ships Project (PMS304).

2. "SESCOMPSPEC2 SESCOMP: Specification on Use and Control; February 1975;" Navy Surface Effect Ships Project (PMS304).

3. "SESCOMPSPEC3 SESCOMP: Specification on Coding Language; February 1975;" Navy Surface Effect Ships Project (PMS304).

4. "SESCOMPSPEC4 SESCOMP: Specification on Organization and Design; February 1975;" Navy Surface Effect Ships Project (PMS304).

5. "SESCOMPSPEC5 SESCOMP: Specification on Programming; February 1975;" Navy Surface Effect Ships Project (PMS304).

6. Cuthbert, J.W. et al., "SESCOMP: A System for Procuring and Controlling Modular Computer Programs," Proceedings of the 18th Annual Meeting of the Engineering Data Management and Computer Aided Design Technology Sections of the American Defense Preparedness Association (May 1976).

7. Culpepper, L.M., "A System for Reliable Engineering Software," David W. Taylor Naval Ship Research and Development Report 4588 (Nov 1974).

8. American National Standards Institute, "USA Standard FORTRAN, X3.9-1966."

9. Allen, F.E., "Control Flow Analysis," Proceedings of a Symposium on Compiler Optimization, ACM SIGPLAM Notices, 5,7 (Jul 1970).

10. Kennedy, K., "A Global Flow Analysis Algorithm," Inter-Journal of Computer Math., Sec. A, 3 (1971).

11. Berkowitz, S., "Graph Information Retrieval Language; Programming Manual for FORTRAN Complement; Revision One," David W. Taylor Naval Ship Research and Development Center Report 76-0085 February 1976, Bethesda, Maryland 20084.

12. Ramamoorthy, C. et al., "Design and Construction of an Automated Software Evaluation System," Record, 1973 IEEE Symposium on Software Reliability.

13. Zaritsky, I., "GIRS (Graph Information Retrieval System) Users Guide and Implementation," David W. Taylor Naval Ship Research and Development Center Report (not yet published).

14. Culpepper, L.M. and R. Regen, "AUDIT, A System for Software Engineering for the CDC 6000," David W. Taylor Naval Ship Research and Development Report 4587 (Nov 1974).

APPENDIX A

SAMPLE INPUT/OUTPUT

Interface Definition File

```
ABS      1 1 4
102
AERODA   8 0 1
100 100 100 100 100 100 402 402
AERODI   3 0 7       21
      2 1      1 4     18 1
AEROSA   9 0 1
102 100 100 100 100 100 100 402 402
AEROSI   2 0 7       8
      7 1      1 4
AIMAG    1 1 4
202
AINT     1 1 4
102
ALOG     1 1 4
102
ALOG10   1 1 4
102
AMAX0    -1 1 4
402
AMAX1    -1 1 4
102
AMIN0    -1 1 4
402
AMIN1    -1 1 4
102
AMOD     2 1 4
102 102
APNDGA   8 0 1
110 110 110 110 110 110 402 402
APNDGC   1 0 7       60
     60 1
APPNDG   3 0 7     262
      1 4    200 1     61 4
ASEDV    5 1 2
102 102 402 421 402
ASEDV1   0 0 8
ASELG    4 1 2
102 402 421 402
ASELG1   0 0 8
ASEL1    0 0 8
ASEL10   4 1 2
102 402 421 402
ASERT    4 1 2
102 402 421 402
ASERT1   0 0 8
ATAN     1 1 4
102
ATAN2    2 1 4
102 102
BARVAL   1 0 7       3
      3 1
BINBLK   1 0 1
402
BMCO     2 0 7       2
      1 1      1 4
BOWSLA   9 0 1
100 100 100 100 100 100 100 402 402
BOWSLI   3 0 7       59
      1 4      56 1      2 4
BOWSLC   1 0 7       40
     40 1
```

PRECEDING PAGE BLANK-NOT FILMED

```
CABS     1 1 4
202
CCOS     1 2 4
202
CEXP     1 2 4
202
CGLOC    1 0 7       2
      2 1
CLOG     1 2 4
202
CMPLX    2 2 4
102 102
CCLFLA   0 0 3
CCLUMN   1 0 7       2
      2 4
CONINP   5 0 7      38
      2 1      2 4     11 1      1 4     22 1
CONJG    1 2 4
202
COS      1 1 4
102
CSIN     1 2 4
202
CSQRT    1 2 4
202
DABS     1 3 4
302
DATAN    1 3 4
302
DATAN2   2 3 4
302 302
DBLE     1 3 4
102
DCOS     1 3 4
302
DEXP     1 3 4
302
DIM      2 1 4
102 102
DLOG     1 3 4
302
DLOG10   1 3 4
302
DMAX1   -1 3 4
302
DMIN1   -1 3 4
302
DMOD     2 3 4
302 302
DSEDV    5 3 2
302 302 402 421 402
DSEDV1   0 0 8
DSELG    4 3 2
302 402 421 402
DSELG1   0 0 8
DSEL1    0 0 8
DSEL10   4 3 2
302 402 421 402
DSERT    4 3 2
302 402 421 402
DSERT1   0 0 8
DSIGN    2 3 4
302 302
DSIN     1 3 4
302
DSQRT    1 3 4
302
```

```
DUMB     1 0 7       1
      1 4
ENGINA   3 0 3
102 402 402
ENGINI   8 0 7    1507
      8 4    280 1      7 4      7 1     21 4   1176 1      1 4      7 1
EQNCO    2 0 7      41
      1 4      40 1
EXP      1 1 4
102
FANA     7 0 1
102 112 110 100 100 402 402
FANDYA   3 0 3
110 402 402
FANI     5 0 7     354
     21 4      80 1     45 4    205 1      3 4
FG1A     6 1 2
102 402 112 112 401 402
FG2A    10 1 2
102 102 402 402 112 112 122 401 401 402
FLAGS    1 0 7       4
      4 4
FLOAT    1 1 4
402
FROUDE   1 0 7       2
      2 1
FTNBIN   3 0 0
402 402 412
GBOW     1 0 7       3
      3 1
HELMS    3 0 7     409
      8 4    400 1      1 4
IABS     1 4 4
402
IDIM     2 4 4
402 402
IDINT    1 4 4
302
IFIX     1 4 4
102
INCONA   3 0 3
100 400 400
INOUT    0 0 6
INT      1 4 4
102
INTGRA   7 0 3
402 112 102 112 402 401 402
ISEDV    5 4 2
402 402 402 421 402
ISEDV1   0 0 8
ISEGO    5 4 2
402 402 402 421 402
ISIGN    2 4 4
402 402
LEAKER   1 0 7       1
      1 1
LOADS    1 0 7       1
      1 1
MASSES   2 0 7    1006
      1 4   1005 1
MATRIX   1 0 7      36
     36 1
MAX0    -1 4 4
402
MAX1    -1 4 4
102
```

```
MINVA    6 0 1
121 402 100 410 410 402
MINO    -1 4 4
402
MIN1    -1 4 4
102
MOD      2 4 4
402 402
MSIDW    1 0 7      25
     25 1
MWAVE    1 0 7       6
      6 1
OPTION   1 0 7       3
      3 4
PAGE1    0 0 8
PHYCON   1 0 7       7
      7 1
PRINT    1 0 7       3
      3 4
PROPA    8 0 1
100 100 100 100 100 100 402 402
PROPI    2 0 7       8
     7 1      1 4
REAL     1 1 4
202
RMSA     3 0 3
112 110 402
ROLCOL   5 0 1
401 422 402 412 402
SAM      7 0 1
402 402 102 102 102 402 402
SEG01    0 0 8
SESA     0 0 5
SESBT    4 0 1
402 422 402 402
SESCOM   2 0 7      25
     13 0     12 4
SESPL1   3 0 1
402 402 402
SIDTAB   0 0 5
SIDWLA  12 0 1
102 402 100 100 100 100 100 100 100 402 102 402
SIDWLI   3 0 7      27
      1 4     25 1      1 4
SIGN     2 1 4
102 102
SIN      1 1 4
102
SLOPE    1 0 7       2
      2 1
SNGL     1 1 4
302
SPRAY    4 0 7      55
      1 4      5 1      1 4     48 1
SPRDRG   1 0 7       1
      1 1
SQRT     1 1 4
102
START    0 0 6
STNSLA   9 0 1
100 100 100 100 100 100 100 402 402
STNSLI   3 0 7      30
      1 4     28 1      1 4
STNSLO   1 0 7      40
     40 1
```

```
TAN      1 1 4
102
TANH     1 1 4
102
TIMES    1 0 7       5
      5 1
TORQUE   0 0 9
TRACE1   0 0 8
TRAP     2 1 2
112 402
UNITS    1 0 7       9
      9 4
USED01   0 0 8
USED02   0 0 8
USED03   0 0 8
USED04   0 0 8
USED06   0 0 8
USED08   0 0 8
USED10   0 0 8
USED11   0 0 8
USED14   0 0 8
USED17   0 0 8
USED18   0 0 8
USED19   0 0 8
USED20   0 0 8
VARBLE   1 0 7      15
     15 1
WAVE     1 0 7       6
      6 1
WAVEA   11 0 1
402 112 112 110 110 110 110 110 110 402 402
WAVESI   4 0 7      79
      1 1      1 4     76 1      1 4
WAVTAB   8 0 7   14050
      1 4      2 1      1 4      2 1      1 4      2 1      1 4 14040 1
XYZTAB   1 0 7     851
    851 1
```

Audit and Flow Analysis of a Main Program

```
C     OPTION 1 SAMPLE MAIN PROGRAM....SESCOMP.US-02-060176
C     CATEGORY 1 LABELED COMMON
C       .
C       .
       COMMON/SESCOM/CASE(13),INA,INB,INC,IOX,NPAGX,LINX,IOY,NPAGY,LINY, IO7,NPAG7,LIN7
C       .
C       .
C     CATEGORY 2 LABELED COMMON
C       .
C       .
C     CATEGORY 3 LABELED COMMON
C       .
C       .
     1 FORMAT(13A4,I4)
     2 FORMAT(5X,52HOPTION 1 SAMPLE MAIN PROGRAM....SESCOMP.US-02-060176)
     3 FORMAT(I4)
C     CASE LOGIC
       CALL INOUT
       CALL INOUT
       READ(INA,3) NCASE
```

```
GRAPH HAS BEEN PLACED INTO MEMORY


           DO 4 I=1,NCASE

           READ(INA,1) CASE,MODE

           CALL START
           CALL START

           CALL XAMPL(IERR,1)
**********************************************************************
     WARNING - THIS MODULE IS NOT IN THE SESCOMP LIST
**********************************************************************
           CALLXAMPL(IERR,1)

           CALL SESPL1(IOX,1,2)
           CALLSESPL1(IOX,1,2)

           WRITE(IOX,2)

           CALL XAMPL(IERR,MODE)
           CALLXAMPL(IERR,MODE)

           IF(IERR.EQ.0) GO TO 5

           CALL INOUT
           CALL INOUT

         4 CONTINUE

         5 STOP

           END
```

SYMBOL TABLE FOR MODULE MAIN

VARIABLES

NAME	TYPE		RELOCATION
CASE	REAL	ARRAY 1	SESCOM
INA	INTEGER		SESCOM
IOx	INTEGER		SESCOM
NCASE	INTEGER		
I	INTEGER		
MODE	INTEGER		
IERR	INTEGER		

EXTERNALS

NAME	TYPE	ARGS
INOUT		0
START		0
XAMPL		2
SESPL1		3

STATEMENT LABELS

1 2 3 4 5

COMMON BLOCKS

NAME	LENGTH
SESCOM	25

****** RESULTS OF FLOW ANALYSIS ******

NO ERRORS FOUND

NUMBER OF PATHS CHECKED- 2

Audit, Flow Analysis, and Roll Call (Mode=2) of a Subroutine Module

```
      SUBROUTINE APNDGA (FX,FY,FZ,FK,FM,FN,ION,MODE)
C     ******************   CATEGORY 1 COMMON BLOCKS   ******************
      COMMON /APNDGO/ XAPPF(20),YAPPF(20),ZAPPF(20)
      COMMON /APPNDG/ NAPP,XAPP(20),YAPP(20),ZAPP(20),CHOT(20),CHOR(20),SPAN(20),TOVC(20),ANGINC(20)
,ANGCNT(20),ANGSWP(20),ILIFT(20),IATT(20),ITYPE(20),IAPNDG
      COMMON /CGLOC/ XS,ZS
      COMMON /FLAGS/ IJOB,ICASE,ISTEP,IERR
      COMMON /PHYCON/ G,RHO,HRHO,ENU,RHOINF,PINF,GAM
      COMMON /SESCOM/ CASE(13),INA,INB,INC,IOX,NPAGX,LINX,IOY,NPAGY,LINY,IOZ,NPAGZ,LINZ
      COMMON /VARBLE/ VAL(15)
C     ******************   CATEGORY 2 COMMON BLOCKS   ******************
      COMMON /USED03/ MUSE
C     ---------------------------------------------------------------
1     FORMAT (4X,52HSUBROUTINE APNDGA.............DRAFT.  01-01-051474)
C     ---------------------------------------------------------------
2     FORMAT (12H ***ERROR***)
3     FORMAT (5X,12HUSE INDEX IS,I3,21H CHECK INITIALIZATION)
4     FORMAT (5X,24H IMPROPER MODE INDEX OF ,I4,13H ENCOUNTERED./5X,39H ZERO SUBSTITUTED, ROLL CALL
CONTINUES.)
5     FORMAT (5X,5HUSED ,I5,40H TIMES WITH MODE INDEX GREATER THAN ZERO)
6     FORMAT (/8H APNDGA /120H APNDG  ANG INC  CANT ANG      X-FORCE        Y-FORCE         Z-FORCE
```

```
      ROLL MOM.        PITCH MOM.        YAW MOM. /120H  NO.     DEG.       DEG.          LBS.
   LBS.              LBS.              FT-LBS.          FT-LBS.          FT-LBS. /)


7     FORMAT (I4,2F10.2,6E16.4)

      DIMENSION FX(1), FY(1), FZ(1), FK(1), FM(1), FN(1)

      DIMENSION RCOS(20), RSIN(20)

      DIMENSION ETA(20), VWAVE(20), WWAVE(20), DUM(20)

      EQUIVALENCE (VAL(2),U), (VAL(3),V), (VAL(4),W), (VAL(5),P), (VAL(6),Q), (VAL(7),R), (VAL(8),PH
   I), (VAL(9),THETA), (VAL(10),Z)

C     CHECK MODE

      IF (MODE.LT.1) GO TO 17
GRAPH HAS BEEN PLACED INTO MEMORY



      MUSE=MUSE+1
      MUSE=MUSE+1

C     BEGINNING OF JOB LOGIC

      IF (IJOB.EQ.1) GO TO 8

      PI=4.*ATAN(1.)
      PI=4.*ATAN(1.)

      PI8=PI/8.
      PI8=PI/8.

      RAD=180.0/PI
      RAD=180.0/PI

8     CONTINUE

C     BEGINNING OF CASE LOGIC

      IF (ICASE.EQ.1) GO TO 10

      DO 9 I=1,NAPP

      RSIN(I)=SIN(ANGCNT(I))
      RSIN(I)=SIN(ANGCNT(I))
```

```
9       RCOS(I)=COS(ANGCNT(I))
9       RCOS(I)=COS(ANGCNT(I))

10      CONTINUE

        Q2=U*U*HRHO
        Q2=U*U*HRHO

C       BEGINNING OF TIME STEP LOGIC

        IF (ISTEP.EQ.1) GO TO 11

        CALL WAVEA (NAPP,XAPP,YAPP,ETA,DUM,DUM,DUM,VWAVE,WWAVE,ION,MODE)
        CALL ROLCHK (1HW,1HA,1HV,1HE,1HA,1H )

11      CONTINUE

        DO 14 I=1,NAPP

C       CALCULATE WETTED SPAN

        DSR=Z+ZS-XAPP(I)*THETA+YAPP(I)*PHI+ETA(I)
        DSR=Z+ZS-XAPP(I)*THETA+YAPP(I)*PHI+ETA(I)

        RSPAN=SPAN(I)
        RSPAN=SPAN(I)

        RAREA=.5*(CHOT(I)+CHOR(I))*RSPAN
        RAREA=.5*(CHOT(I)+CHOR(I))*RSPAN

        ENDFAC=1.0
        ENDFAC=1.0

        DELT=DSR-(ZS-ZAPP(I))
        DELT=DSR-(ZS-ZAPP(I))

C       IS APPENDAGE OUT OF WATER

        IF (DELT.GE.0.0) GO TO 13

C       IS APPENDAGE HORIZONTAL OR CANTED UPWARD

        IF (RCOS(I).LE.0.0) GO TO 12

        RSPAN=RSPAN+DELT/RCOS(I)
        RSPAN=RSPAN+DELT/RCOS(I)
```

```
      IF (RSPAN.LE.0.0) GO TO 12

      RAREA=.5*(CHDT(I)+CHDR(I)*RSPAN/SPAN(I))*RSPAN
      RAREA=.5*(CHDT(I)+CHDR(I)*RSPAN/SPAN(I))*RSPAN

      ENDFAC=1.0
      ENDFAC=1.0

      GO TO 13

12    FX(I)=0.0
12    FX(I)=0.0

      FY(I)=0.0
      FY(I)=0.0

      FZ(I)=0.0
      FZ(I)=0.0

      FK(I)=0.0
      FK(I)=0.0

      FM(I)=0.0
      FM(I)=0.0

      FN(I)=0.0
      FN(I)=0.0

      GO TO 14

C     CALCULATE LIFT

13    VH=V+XAPP(I)*R-ZAPP(I)*P-VWAVE(I)
13    VH=V+XAPP(I)*R-ZAPP(I)*P-VWAVE(I)

      VV=W-XAPP(I)*Q+YAPP(I)*P+U*THETA-WWAVE(I)
      VV=W-XAPP(I)*Q+YAPP(I)*P+U*THETA-WWAVE(I)

      PHIV=ANGCNT(I)+PHI
      PHIV=ANGCNT(I)+PHI

      VN=SQRT(VV*VV+VH*VH)
      VN=SQRT(VV*VV+VH*VH)

      PHIN=PHIV
      PHIN=PHIV
```

```
      IF (VN.NE.0.0) PHIN=PHIV+ATAN2(VV,VH)

      VN=VN*COS(PHIN)
      VN=VN*COS(PHIN)

      ALPHAI=ATAN2(VN,U)
      ALPHAI=ATAN2(VN,U)

      EFFANG=ANGINC(I)-ALPHAI
      EFFANG=ANGINC(I)-ALPHAI

      QQ=Q2*RAREA
      QQ=Q2*RAREA

      REY=U*(RAREA/RSPAN)/ENU
      REY=U*(RAREA/RSPAN)/ENU

      CFR=.427/(ALOG10(REY)-.407)**2.64
      CFR=.427/(ALOG10(REY)-.407)**2.64

      RASPR=RSPAN*RSPAN/RAREA
      RASPR=RSPAN*RSPAN/RAREA

      RCLB=2.*PI*RASPR/(RASPR+3.)
      RCLB=2.*PI*RASPR/(RASPR+3.)

      FLIFT=QQ*ENDFAC*RCLB*EFFANG
      FLIFT=QQ*ENDFAC*RCLB*EFFANG

      CD=2.*CFR+PI8*TOVC(I)*TOVC(I)*(1.+G*RSPAN/(U*U))+RCLB*EFFANG*EFFANG

      CD=2.*CFR+PI8*TOVC(I)*TOVC(I)*(1.+G*RSPAN/(U*U))+RCLB*EFFANG*EFFAN
     G

C     CALCULATE MOMENT ARMS

      XAPPF(I)=XAPP(I)
      XAPPF(I)=XAPP(I)

      YAPPF(I)=YAPP(I)-(SPAN(I)-.5*RSPAN)*RSIN(I)
      YAPPF(I)=YAPP(I)-(SPAN(I)-.5*RSPAN)*RSIN(I)

      ZAPPF(I)=ZAPP(I)+(SPAN(I)-.5*RSPAN)*RCOS(I)+YAPPF(I)*PHI-XAPPF(I)*THETA

      ZAPPF(I)=ZAPP(I)+(SPAN(I)-.5*RSPAN)*RCOS(I)+YAPPF(I)*PHI-XAPPF(I)*
     THETA

      FX(I)=-CD*QQ
      FX(I)=-CD*QQ
```

```
      FY(I)=FLIFT*RCOS(I)
      FY(I)=FLIFT*RCOS(I)

      FZ(I)=FLIFT*RSIN(I)
      FZ(I)=FLIFT*RSIN(I)

      FK(I)=-ZAPPF(I)*FY(I)+YAPPF(I)*FZ(I)
      FK(I)=-ZAPPF(I)*FY(I)+YAPPF(I)*FZ(I)

      FM(I)=ZAPPF(I)*FX(I)-XAPPF(I)*FZ(I)
      FM(I)=ZAPPF(I)*FX(I)-XAPPF(I)*FZ(I)

      FN(I)=-YAPPF(I)*FX(I)+XAPPF(I)*FY(I)
      FN(I)=-YAPPF(I)*FX(I)+XAPPF(I)*FY(I)

14    CONTINUE

      IF (ION.NE.IAPNOG) RETURN

      ITEMP=5+NAPP
      ITEMP=5+NAPP

      CALL SESPL1 (IOX,ITEMP,MODE)
      CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)

      WRITE (IOX,6)

      DO 15 I=1,NAPP

      ANG1=ANGINC(I)*RAD
      ANG1=ANGINC(I)*RAD

      ANG2=ANGCNT(I)*RAD
      ANG2=ANGCNT(I)*RAD

      WRITE (IOX,7) I,ANG1,ANG2,FX(I),FY(I),FZ(I),FK(I),FM(I),FN(I)

15    CONTINUE

      RETURN

C               START SESCOM ROLL CALL CODING WITHOUT BUFFER TRACING

C      PATH FOR ERRORS

16    CALL SESPL1 (IOX,3,1)
16    CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)
```

```
      WRITE (IOX,2)

      WRITE (IOX,1)

      WRITE (IOX,3) MUSE

      GO TO 24

C      ROLL CALL CODING

17    NMOD=MODE
17    NMOD=MODE

      IF (MUSE) 16,24,18

18    IF (MODE.GE.-3) GO TO 19

      IF (MODE.GE.-7) GO TO 20

      IO=IOZ
      IO=IOZ

      IF (MODE.GE.-11) GO TO 21

19    IO=IOX
19    IO=IOX

      IF (MODE.GE.-11) GO TO 21

      CALL SESPL1 (IO,4,1)
      CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)

      WRITE (IO,2)

      WRITE (IO,1)

      WRITE (IO,4) MODE

      NMOD=0
      NMOD=0

      GO TO 21

20    IO=IOY
20    IO=IOY
```

```
21    CALL SESPL1 (IO,1,1)
21    CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)

      WRITE (IO,1)

      NMODE=ISIGN(NMOD,1)+1
      NMODE=ISIGN(NMOD,1)+1

      INDX=MOD(NMODE,4)+1
      INDX=MOD(NMODE,4)+1

C       GO TO SELECTED MODE

      GO TO (23,22,23,22), INDX

22    CALL SESPL1 (IO,1,1)
22    CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)

      WRITE (IO,5) MUSE

23    MUSE=0
23    MUSE=0

C     CALL REFERENCED MODULES IN ROLL CALL MODE

      CALL SESPL1 (IO,1,MODE)
      CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)

      CALL WAVEA (DM,DM,DM,DM,DM,DM,DM,DM,DM,DM,MODE)
**********************************************************************
      ARGUMENT NO.  1 HAS INCORRECT TYPE
**********************************************************************
**********************************************************************
      WARNING - ARGUMENT NO.  2 MAY HAVE INCORRECT DIMENSIONALITY
**********************************************************************
**********************************************************************
      WARNING - ARGUMENT NO.  3 MAY HAVE INCORRECT DIMENSIONALITY
**********************************************************************
**********************************************************************
      WARNING - ARGUMENT NO.  4 MAY HAVE INCORRECT DIMENSIONALITY
**********************************************************************
**********************************************************************
      WARNING - ARGUMENT NO.  5 MAY HAVE INCORRECT DIMENSIONALITY
**********************************************************************
**********************************************************************
      WARNING - ARGUMENT NO.  6 MAY HAVE INCORRECT DIMENSIONALITY
**********************************************************************
**********************************************************************
      WARNING - ARGUMENT NO.  7 MAY HAVE INCORRECT DIMENSIONALITY
**********************************************************************
**********************************************************************
      WARNING - ARGUMENT NO.  8 MAY HAVE INCORRECT DIMENSIONALITY
**********************************************************************
**********************************************************************
```

```
      WARNING - ARGUMENT NO.  9 MAY HAVE INCORRECT DIMENSIONALITY
**********************************************************************
**********************************************************************
      ARGUMENT NO. 10 HAS INCORRECT TYPE
**********************************************************************
           CALL ROLCHK (1HW,1HA,1HV,1HE,1HA,1H )

C                  END SESCOM ROLL CALL CODING

24    RETURN

      END
```

SYMBOL TABLE FOR MODULE APNDGA

VARIABLES

NAME	TYPE		RELOCATION
FX	REAL	ARRAY 1	F. P.
FY	REAL	ARRAY 1	F. P.
FZ	REAL	ARRAY 1	F. P.
FK	REAL	ARRAY 1	F. P.
FM	REAL	ARRAY 1	F. P.
FN	REAL	ARRAY 1	F. P.
ION	INTEGR		F. P.
MODE	INTEGR		F. P.
XAPPF	REAL	ARRAY 1	APNDGO
YAPPF	REAL	ARRAY 1	APNDGO
ZAPPF	REAL	ARRAY 1	APNDGO
NAPP	INTEGR		APPNDG
XAPP	REAL	ARRAY 1	APPNDG
YAPP	REAL	ARRAY 1	APPNDG
ZAPP	REAL	ARRAY 1	APPNDG
CHDT	REAL	ARRAY 1	APPNDG
CHDK	REAL	ARRAY 1	APPNDG
SPAN	REAL	ARRAY 1	APPNDG
TOVC	REAL	ARRAY 1	APPNDG
ANGINC	REAL	ARRAY 1	APPNDG
ANGCNT	REAL	ARRAY 1	APPNDG
IAPNDG	INTEGR		APPNDG
ZS	REAL		CGLOC
IJOB	INTEGR		FLAGS
ICASE	INTEGR		FLAGS
ISTEP	INTEGR		FLAGS
G	REAL		PHYCON
HRHO	REAL		PHYCON
ENU	REAL		PHYCON
IOX	INTEGR		SESCOM
IOY	INTEGR		SESCOM
IOZ	INTEGR		SESCOM
VAL	REAL	ARRAY 1	VARBLE
MUSE	INTEGR		USED03
RCOS	REAL	ARRAY 1	
RSIN	REAL	ARRAY 1	
ETA	REAL	ARRAY 1	
VWAVE	REAL	ARRAY 1	
HWAVE	REAL	ARRAY 1	
DUM	REAL	ARRAY 1	

U	REAL
V	REAL
W	REAL
P	REAL
Q	REAL
R	REAL
PHI	REAL
THETA	REAL
Z	REAL
PI	REAL
PI8	REAL
RAD	REAL
I	INTEGR
Q2	REAL
DSR	REAL
RSPAN	REAL
RAREA	REAL
ENDFAC	REAL
DELT	REAL
VH	REAL
VV	REAL
PHIV	REAL
VN	REAL
PHIN	REAL
ALPHAI	REAL
EFFANG	REAL
QQ	REAL
REY	REAL
CFR	REAL
RASPR	REAL
RCL8	REAL
FLIFT	REAL
CD	REAL
ITEMP	INTEGR
ANG1	REAL
ANG2	REAL
NMOD	INTEGR
IO	INTEGR
NMODE	INTEGR
INDX	INTEGR
DM	REAL

EXTERNALS

NAME	TYPE	ARGS
ATAN	REAL	1
SIN	REAL	1
COS	REAL	1
WAVEA		11
SQRT	REAL	1
ATAN2	REAL	2
ALOG10	REAL	1
SESPL1		3
ISIGN	INTEGR	2
MOD	INTEGR	2

STATEMENT LABELS

1	2	3	4	5
6	7	17	8	10
9	11	14	13	12
15	16	24	18	19
20	21	23	22	

COMMON BLOCKS

NAME	LENGTH
APNDGO	60
APPNDG	262
CGLOC	2
FLAGS	4
PHYCON	7
SESCOM	25
VARBLE	15
USED03	1

```
****** RESULTS OF FLOW ANALYSIS ******

**********************************************************************************
      THE VARIABLE DM      IS REFERENCED BUT NOT DEFINED
      ALONG THE PATH    17    18    19    21    23
**********************************************************************************
      THE VARIABLE ETA     IS REFERENCED BUT NOT DEFINED
      ALONG THE PATH     8    10    11
**********************************************************************************
      THE VARIABLE VWAVE   IS REFERENCED BUT NOT DEFINED
      ALONG THE PATH     8    10    11    13
**********************************************************************************
      THE VARIABLE WWAVE   IS REFERENCED BUT NOT DEFINED
      ALONG THE PATH     8    10    11    13
**********************************************************************************
      THE VARIABLE PI      IS REFERENCED BUT NOT DEFINED
      ALONG THE PATH     8    10    11    13
**********************************************************************************
      THE VARIABLE PI8     IS REFERENCED BUT NOT DEFINED
      ALONG THE PATH     8    10    11    13
**********************************************************************************
      THE VARIABLE RSIN    IS REFERENCED BUT NOT DEFINED
      ALONG THE PATH     8    10    11    13
**********************************************************************************
      THE VARIABLE RCOS    IS REFERENCED BUT NOT DEFINED
      ALONG THE PATH     8    10    11    13
**********************************************************************************
      THE VARIABLE RAD     IS REFERENCED BUT NOT DEFINED
      ALONG THE PATH     8    10    11    13    14



      NUMBER OF PATHS CHECKED-     78

                                        FLOW ANALYSIS TOOK     1.394 CP SECONDS
```

GLOBAL REFERENCE TABLE

EXTERNAL REFERENCES

ATAN	ANSI FUNCTION
SIN	ANSI FUNCTION
COS	ANSI FUNCTION
WAVEA	SUBROUTINE MODULE
SQRT	ANSI FUNCTION
ATAN2	ANSI FUNCTION
ALOG10	ANSI FUNCTION
SESPL1	SUBROUTINE MODULE
ISIGN	ANSI FUNCTION
MOD	ANSI FUNCTION

LABELLED COMMON BLOCKS

BLOCK NAME	SIZE	CLASS	
APNDGO	60	CATEGORY	1
APPNDG	262	CATEGORY	1
CGLOC	2	CATEGORY	1
FLAGS	4	CATEGORY	1
PHYCON	7	CATEGORY	1
SESCOM	25	CATEGORY	1
VARBLE	15	CATEGORY	1
USED03	1	CATEGORY	2

SUBROUTINES ENCOUNTERED

APNDGA	SUBROUTINE MODULE

```
      SUBROUTINE APNDGA (FX,FY,FZ,FK,FM,FN,ION,MODE)
      COMMON /APNDGO/ XAPPF(20),YAPPF(20),ZAPPF(20)
      COMMON /APPNDG/ NAPP,XAPP(20),YAPP(20),ZAPP(20),CHDT(20),CHDR(20),
     *SPAN(20),IOVC(20),ANGINC(20),ANGCNT(20),ANGSWP(20),ILIFT(20),IATT(
     *20),ITYPE(20),IAPNDG
      COMMON /CGLOC/ XS,ZS
      COMMON /FLAGS/ IJOB,ICASE,ISTEP,IERR
      COMMON /PHYCON/ G,RHO,HRHO,ENU,RHOINF,PINF,GAM
      COMMON /SESCOM/ CASE(13),INA,INB,INC,IOX,NPAGX,LINX,IOY,NPAGY,LINY
     *,IOZ,NPAGZ,LINZ
      COMMON /VARBLE/ VAL(15)
      COMMON /USED03/ MUSE
1     FORMAT (4X,52HSUBROUTINE APNDGA.............DRAFT.  01-01-051474
     *)
2     FORMAT (12H ***ERROR***)
3     FORMAT (5X,12HUSE INDEX IS,I3,21H CHECK INITIALIZATION)
4     FORMAT (5X,24H IMPROPER MODE INDEX OF ,I4,13H ENCOUNTERED./5X,39H
     *ZERO SUBSTITUTED, ROLL CALL CONTINUES.)
5     FORMAT (5X,5HUSED ,I5,40H TIMES WITH MODE INDEX GREATER THAN ZERO)
6     FORMAT (/8H APNDGA /120H APNDG  ANG INC  CANT ANG      X-FORCE
     *    Y-FORCE          Z-FORCE          ROLL MOM.        PITCH MOM.
     *   YAW MOM. /120H  NO.    DEG.       DEG.          LBS.             L
     *BS.             LBS.             FT-LBS.         FT-LBS.           FT-
     *LBS. /)
7     FORMAT (I4,2F10.2,6E16.4)
      DIMENSION FX(1), FY(1), FZ(1), FK(1), FM(1), FN(1)
      DIMENSION RCOS(20), RSIN(20)
      DIMENSION ETA(20), WWAVE(20), WWAVE(20), DUM(20)
      EQUIVALENCE (VAL(2),U), (VAL(3),V), (VAL(4),W), (VAL(5),P), (VAL(6
     *),Q), (VAL(7),R), (VAL(8),PHI), (VAL(9),THETA), (VAL(10),Z)
      IF (MODE.LT.1) GO TO 17
      MUSE=MUSE+1
      IF (IJOB.EQ.1) GO TO 8
      PI=4.*ATAN(1.)
      PI8=PI/8.
      RAD=180.0/PI
8     CONTINUE
      IF (ICASE.EQ.1) GO TO 10
      DO 9 I=1,NAPP
      RSIN(I)=SIN(ANGCNT(I))
9     RCOS(I)=COS(ANGCNT(I))
10    CONTINUE
      Q2=U*U*HRHO
      IF (ISTEP.EQ.1) GO TO 11
      CALL ROLCHK (1HW,1HA,1HV,1HE,1HA,1H )
11    CONTINUE
      DO 14 I=1,NAPP
      DSR=Z+ZS-XAPP(I)*THETA+YAPP(I)*PHI+ETA(I)
      RSPAN=SPAN(I)
      RAREA=.5*(CHDT(I)+CHDR(I))*RSPAN
      ENDFAC=1.0
      DELT=DSR-(ZS-ZAPP(I))
      IF (DELT.GE.0.0) GO TO 13
      IF (RCOS(I).LE.0.0) GO TO 12
      RSPAN=RSPAN+DELT/RCOS(I)
      IF (RSPAN.LE.0.0) GO TO 12
      RAREA=.5*(CHDT(I)+CHDR(I)*RSPAN/SPAN(I))*RSPAN
```

```
      ENDFAC=1.0
      GO TO 13
12    FX(I)=0.0
      FY(I)=0.0
      FZ(I)=0.0
      FK(I)=0.0
      FM(I)=0.0
      FN(I)=0.0
      GO TO 14
13    VH=V+XAPP(I)*R-ZAPP(I)*P-VWAVE(I)
      VV=W-XAPP(I)*Q+YAPP(I)*P+U*THETA-WWAVE(I)
      PHIV=ANGCNT(I)+PHI
      VN=SQRT(VV*VV+VH*VH)
      PHIN=PHIV
      IF (VN.NE.0.0) PHIN=PHIV-ATAN2(VV,VH)
      VN=VN*COS(PHIN)
      ALPHAI=ATAN2(VN,U)
      EFFANG=ANGINC(I)-ALPHAI
      QQ=Q2*RAREA
      REY=U*(RAREA/RSPAN)/ENU
      CFR=.427/(ALOG10(REY)-.407)**2.64
      RASPR=RSPAN*RSPAN/RAREA
      RCLB=2.*PI*RASPR/(RASPR+3.)
      FLIFT=QQ*ENDFAC*RCLB*EFFANG
      CD=2.*CFR+PI8*TOVC(I)*TOVC(I)*(1.+G*RSPAN/(U*U))+RCLB*EFFANG*EFFAN
     *G
      XAPPF(I)=XAPP(I)
      YAPPF(I)=YAPP(I)-(SPAN(I)-.5*RSPAN)*RSIN(I)
      ZAPPF(I)=ZAPP(I)+(SPAN(I)-.5*RSPAN)*RCOS(I)+YAPPF(I)*PHI-XAPPF(I)*
     *THETA
      FX(I)=-CD*QQ
      FY(I)=FLIFT*RCOS(I)
      FZ(I)=FLIFT*RSIN(I)
      FK(I)=-ZAPPF(I)*FY(I)+YAPPF(I)*FZ(I)
      FM(I)=ZAPPF(I)*FX(I)-XAPPF(I)*FZ(I)
      FN(I)=-YAPPF(I)*FX(I)+XAPPF(I)*FY(I)
14    CONTINUE
      IF (ION.NE.IAPNOG) RETURN
      ITEMP=5+NAPP
      CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)
      WRITE (IOX,6)
      DO 15 I=1,NAPP
      ANG1=ANGINC(I)*RAD
      ANG2=ANGCNT(I)*RAD
      WRITE (IOX,7) I,ANG1,ANG2,FX(I),FY(I),FZ(I),FK(I),FM(I),FN(I)
15    CONTINUE
      RETURN
16    CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)
      WRITE (IOX,2)
      WRITE (IOX,1)
      WRITE (IOX,3) MUSE
      GO TO 24
17    NMOD=MODE
      IF (MUSE) 16,24,18
18    IF (MODE.GE.-3) GO TO 19
      IF (MODE.GE.-7) GO TO 20
      IO=IOZ
```

```
      IF (MODE.GE.-11) GO TO 21
19    IO=IOX
      IF (MODE.GE.-11) GO TO 21
      CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)
      WRITE (IO,2)
      WRITE (IO,1)
      WRITE (IO,4) MODE
      NMOD=0
      GO TO 21
20    IO=IOY
21    CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)
      WRITE (IO,1)
      NMODE=ISIGN(NMOD,1)+1
      INDX=MOD(NMODE,4)+1
      GO TO (23,22,23,22), INDX
22    CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)
      WRITE (IO,5) MUSE
23    MUSE=0
      CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)
      CALL ROLCHK (1HW,1HA,1HV,1HE,1HA,1H )
24    RETURN
      END
```

```
      PROGRAM ROLCAL(OUTPUT,TAPE6=OUTPUT,TAPE3,TAPE9,
     * TAPE10,TAPE11,TAPE12,TAPE13,TAPE14,TAPE15)
      COMMON/APNDGO/IX 0(     60)
      COMMON/APPNDG/IX 1(    262)
      COMMON/CGLOC /IX 2(      2)
      COMMON/FLAGS /IX 3(      4)
      COMMON/PHYCON/IX 4(      7)
      COMMON/SESCOM/IX 5(     25)
      COMMON/VARBLE/IX 6(     15)
      COMMON/USED03/IX 7(      1)
      J=1
      MODE=2
      REWIND 13
      REWIND 14
      REWIND 15
      DO 10 I=1,13
      J=J-1
      DO 1000 K=1,     60
      IX 0(K)=1
 1000 CONTINUE
      DO 1001 K=1,    262
      IX 1(K)=1
 1001 CONTINUE
      DO 1002 K=1,      2
      IX 2(K)=1
 1002 CONTINUE
      DO 1003 K=1,      4
      IX 3(K)=1
 1003 CONTINUE
      DO 1004 K=1,      7
      IX 4(K)=1
 1004 CONTINUE
      DO 1005 K=1,     25
      IX 5(K)=1
 1005 CONTINUE
      IX 5(17)=10
      IX 5(20)=11
      IX 5(23)=12
      DO 1006 K=1,     15
      IX 6(K)=1
 1006 CONTINUE
      DO 1007 K=1,      1
      IX 7(K)=1
 1007 CONTINUE
      CALL APNDGA(0,0,0,0,0,0,0,J)
      IF(MODE .EQ. 3)GO TO 5
      CALL MODID(J)
    5 ENDFILE 3
   10 CONTINUE
      CALL CMPARE
      REWIND 3
      REWIND 13
      REWIND 14
      REWIND 15
      STOP
      END
```

```
                    RESULTS OF ROLL CALL CHECK

          ALL SUBROUTINES WERE CALLED IN THE ROLL CALL MODE


  OUTPUT DEVICE X

                         MODE INDEX=  0
   SUBROUTINE APNDGA..............DRAFT.  OI-01-051474
    USED     1 TIMES WITH MODE INDEX GREATER THAN ZERO

                         MODE INDEX= -1
   SUBROUTINE APNDGA..............DRAFT.  OI-01-051474

                         MODE INDEX= -2
   SUBROUTINE APNDGA..............DRAFT.  OI-01-051474
    USED     1 TIMES WITH MODE INDEX GREATER THAN ZERO

                         MODE INDEX= -3
   SUBROUTINE APNDGA..............DRAFT.  OI-01-051474

                         MODE INDEX=-12
***ERROR***
   SUBROUTINE APNDGA..............DRAFT.  OI-01-051474
     IMPROPER MODE INDEX OF  -12 ENCOUNTERED.
     ZERO SUBSTITUTED, ROLL CALL CONTINUES.
   SUBROUTINE APNDGA..............DRAFT.  OI-01-051474
    USED     1 TIMES WITH MODE INDEX GREATER THAN ZERO
```

```
 OUTPUT DEVICE Y

                    MODE INDEX= -4
SUBROUTINE APNDGA..............DRAFT.  OI-01-051474
 USED      1 TIMES WITH MODE INDEX GREATER THAN ZERO

                    MODE INDEX= -5
SUBROUTINE APNDGA..............DRAFT.  OI-01-051474

                    MODE INDEX= -6
SUBROUTINE APNDGA..............DRAFT.  OI-01-051474
 USED      1 TIMES WITH MODE INDEX GREATER THAN ZERO

                    MODE INDEX= -7
SUBROUTINE APNDGA..............DRAFT.  OI-01-051474
```

```
 OUTPUT DEVICE Z

                    MODE INDEX= -8
SUBROUTINE APNDGA..............DRAFT.  OI-01-051474
 USED      1 TIMES WITH MODE INDEX GREATER THAN ZERO

                    MODE INDEX= -9
SUBROUTINE APNDGA..............DRAFT.  OI-01-051474

                    MODE INDEX=-10
SUBROUTINE APNDGA..............DRAFT.  OI-01-051474
 USED      1 TIMES WITH MODE INDEX GREATER THAN ZERO

                    MODE INDEX=-11
SUBROUTINE APNDGA..............DRAFT.  OI-01-051474
```

Audit, Flow Analysis, and Roll Call (Mode=3) of a SESCOMP Utility Module

```
      FUNCTION ISEDV(I,N,LOC,NBCT,MODE)
C                  FUNCTION ISEDV.................SESCOMP.US-02-070175
C     ************************************************************
C     *              THIS PROGRAM UNIT IS PROVIDED BY            *
C     *                                                          *
C     *                  DEPARTMENT OF THE NAVY                  *
C     *               SURFACE EFFECT SHIPS PROJECT               *
C     *                     P.O. BOX 34401                       *
C     *                BETHESDA, MARYLAND  20084                 *
C     *                                                          *
C     *  REFERENCE--                                             *
C     *    MODULE VERSION MANUAL - BUFFERED INTEGER DIVISION     *
C     ************************************************************
C*****    ISEDV-A SESCOMP BUFFERED FUNCTION WHICH FINDS THE QUOTIENT
C         OF TWO INTEGERS
C**********I-THE INTEGER DIVIDEND
C**********N-THE INTEGER DIVISOR
C*****    LOC-LOCATION CODE OF FUNCTION REFERENCE IN REFERENCING
C          MODULE. .GT. 0 AND .LE. 10
C*****   NBCT-BUFFER-TRACING ARRAY
```

```
C********MODE=POSITIVE INTEGER INDICATES COMPUTATIONAL MODE
C             .LE. 0 AND .GE. -11 INDICATES ROLL-CALL MODE
C             SEE UTILITY MODULE ROLCOL FOR AN EXPLANATION OF
C             ROLL-CALL MODE VALUES
C  NO BUFFERED FUNCTIONS ARE REFERENCED BY THIS VERSION.
C     CATEGORY 1 COMMON
      COMMON /SESCOM/CASE(13),INA,INB,INC,IOX,NPAGX,LINX,IOY,NPAGY,LINY,IOZ,NPAGZ,LINZ
C*****  CATEGORY 2 COMMON
      COMMON/ISEDV1/NUSE
C*********NUSE-USE-COUNT INDEX
C            >=POSITIVE INTEGER INDICATES NUMBER OF TIMES MODULE HAS
C                                BEEN CALLED WITH A POSITIVE MODE INDEX
C             =0                INDICATES MODULE HAS NOT BEEN CALLED OR
C                                HAS PREVIOUSLY BEEN CALLED IN THE
C                                ROLL-CALL MODE
    1 FORMAT(5X,1344)
    2 FORMAT(12H ***ERROR***)
    3 FORMAT(5X,20H USE-COUNT INDEX IS ,I7,21H CHECK INITIALIZATION)
      DIMENSION NBCT(20,10),MIF(13)
      DATA MIF/4HFUNC,4HTION,4H ISE,4HDV..,4*4H....,4HSESC,4HOMP.,4HUS-0,4H2-07,4H0175/
C*****CHECK MODE
      IF(MODE.LT.1)  GO TO 9988
```

```
GRAPH HAS BEEN PLACED INTO MEMORY


   C*****CHECK FOR NEGATIVE USE COUNT

         IF(NUSE.GE.0) GO TO 10

   C*****PRINT ERROR MESSAGE FOR NEGATIVE USE COUNT

         CALL SESPL1(IOX,3,3)
         CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)

         WRITE(IOX,2)

         WRITE(IOX,1) MIF

         WRITE(IOX,3) NUSE

         NUSE=0
         NUSE=0

      10 NUSE=NUSE+1
      10 NUSE=NUSE+1

         IF(N.NE.0) GO TO 50

         ISEOV=1
         ISEOV=1

         IF(LOC.GT.0.AND.LOC.LE.10) GO TO 20

         IF(LOC.LE.0) GO TO 21

         NBCT(11,10)=NBCT(11,10)+1
         NBCT(11,10)=NBCT(11,10)+1

         GO TO 9999

      21 NBCT(11,1)=NBCT(11,1)+1
      21 NBCT(11,1)=NBCT(11,1)+1

         GO TO 9999

      20 NBCT(11,LOC)=NBCT(11,LOC)+1
      20 NBCT(11,LOC)=NBCT(11,LOC)+1
```

```
      GO TO 9999

   50 ISEOV=I/N
   50 ISEOV=I/N

      GO TO 9999

C***** REFERENCE TO ROLL-CALL UTILITY MODULE

 9988 CALL ROLCOL(NUSE,NBCT,0,MIF,MODE)
 9988 CALL ROLCHK (1HR,1HO,1HL,1HC,1HO,1HL)

C*****CALL REFERENCED MODULES IN THE ROLL-CALL MODE

      CALL SESPL1(IOX,1,MODE)
      CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)

 9999 RETURN

      END
```

SYMBOL TABLE FOR MODULE ISEDV

VARIABLES

NAME	TYPE		RELOCATION
I	INTEGR		F. P.
N	INTEGR		F. P.
LOC	INTEGR		F. P.
NBCT	INTEGR	ARRAY 2	F. P.
MODE	INTEGR		F. P.
IOX	INTEGR		SESCOM
NUSE	INTEGR		ISEDV1
MIF	INTEGR	ARRAY 1	
ISEDV	INTEGR		

EXTERNALS

NAME	TYPE	ARGS
SESPL1		3
ROLCOL		5

STATEMENT LABELS

1	2	3	9988	10
50	20	21	9999	

COMMON BLOCKS

NAME	LENGTH
SESCOM	25
ISEDV1	1

****** RESULTS OF FLOW ANALYSIS ******

NO ERRORS FOUND

NUMBER OF PATHS CHECKED- 9

FLOW ANALYSIS TOOK .038 CP SECONDS

```
                GLOBAL REFERENCE TABLE

              EXTERNAL REFERENCES
        SESPL1      SUBROUTINE MODULE
        ROLCOL      SUBROUTINE MODULE

              LABELLED COMMON BLOCKS
      BLOCK NAME          SIZE          CLASS
        SESCOM              25       CATEGORY  1
        ISEDV1               1       CATEGORY  2


            SUBROUTINES ENCOUNTERED
       ISEDV       FUNCTION MODULE
```

```
      FUNCTION ISEDV(I,N,LOC,NBCT,MODE)
      COMMON /SESCOM/CASE(13),INA,INB,INC,IOX,NPAGX,LINX,IOY,NPAGY,LINY,
     *IOZ,NPAGZ,LINZ
      COMMON/ISEDV1/NUSE
    1 FORMAT(5X,13A4)
    2 FORMAT(12H ***ERROR***)
    3 FORMAT(5X,20H USE-COUNT INDEX IS ,I7,21H CHECK INITIALIZATION)
      DIMENSION NBCT(20,10),MIF(13)
      DATA MIF/4HFUNC,4HTION,4H ISE,4HDV..,4*4H....,4HSESC,4HOMP.,4HUS-0
     *,4H2-07,4H0175/
      IF(MODE.LT.1)  GO TO 9988
      IF(NUSE.GE.0) GO TO 10
      CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)
      WRITE(IOX,2)
      WRITE(IOX,1) MIF
      WRITE(IOX,3) NUSE
      NUSE=0
   10 NUSE=NUSE+1
      IF(N.NE.0) GO TO 50
      ISEDV=1
      IF(LOC.GT.0.AND.LOC.LE.10) GO TO 20
      IF(LOC.LE.0) GO TO 21
      NBCT(11,10)=NBCT(11,10)+1
      GO TO 9999
   21 NBCT(11,1)=NBCT(11,1)+1
      GO TO 9999
   20 NBCT(11,LOC)=NBCT(11,LOC)+1
      GO TO 9999
   50 ISEDV=I/N
      GO TO 9999
 9988 CALL ROLCHK (1HR,1HO,1HL,1HC,1HO,1HL)
      CALL ROLCHK (1HS,1HE,1HS,1HP,1HL,1H1)
 9999 RETURN
      END
```

```
      PROGRAM ROLCAL(OUTPUT,TAPE6=OUTPUT,TAPE3,TAPE9,
     * TAPE10,TAPE11,TAPE12,TAPE13,TAPE14,TAPE15)
      COMMON/SESCOM/IX 0(     25)
      COMMON/ISEDV1/IX 1(      1)
      J=1
      MODE=3
      REWIND 13
      REWIND 14
      REWIND 15
      DO 10 I=1,13
      J=J-1
      DO 1000 K=1,     25
      IX 0(K)=1
 1000 CONTINUE
      IX 0(17)=10
      IX 0(20)=11
      IX 0(23)=12
      DO 1001 K=1,      1
      IX 1(K)=1
 1001 CONTINUE
      CALL ISEDV (0,0,0,0,J)
      IF(MODE .EQ. 3)GO TO 5
      CALL MODID(J)
    5 ENDFILE 3
   10 CONTINUE
      CALL CMPARE
      REWIND 3
      REWIND 13
      REWIND 14
      REWIND 15
      STOP
      END
```

```
 ALL SUBROUTINES WERE CALLED IN THE ROLL CALL MODE
```

Audit and Flow Analysis of Ancillary Subprogram

```
      SUBROUTINE COLFLA
C     ******************   CATEGORY 1 COMMON BLOCKS   ******************
      COMMON /COLUMN/ IVERT,ILATRL
      COMMON /SESCOM/ CASE(13),INA,INB,INC,IOX,NPAGX,LINX,IOY,NPAGY,LINY,IOZ,NPAGZ,LINZ

      COMMON /UNITS/ IS1,IS2,IS3,IS4

1     FORMAT (1H /51X,22HVERTICAL PLANE SUMMARY//10X,4HTIME,7X,8HWAVE AMP,7X,5HDRAFT,9X,5HTHETA,5X,1
0HGAGE PRESS,3X,9HBOW ACCEL,5X,9HC G ACCEL,4X,7HFAN PWR/10X,3HSEC,10X,2HFT,11X,3H FT,11X,3HDEG,10X,3
HPSF,10X,1HG,13X,1HG,11X,2HHP/)

2     FORMAT (7X,F7.2,7X,F6.3,7X,F7.3,2(7X,F6.2),2(7X,F6.3),7X,F7.1)

3     FORMAT (1H /33X,21HLATERAL PLANE SUMMARY//9X,4HTIME,10X,3HPHI,9X,5HBETAS,6X,9HLAT ACCEL,8X,1HU
,6X,11HTURN RADIUS,6X,8HYAW RATE,17H     RUDDER ANGLE/10X,3HSEC,10X,3HDEG,10X,3HDEG,11X,1HG,11X,3HFP
S,10X,2HFT,10X,7HDEG/SEC,10X,3HDEG/)

4     FORMAT (7X,F7.2,2(7X,F6.2),7X,F6.3,7X,F6.2,7X,F6.0,7X,F7.3,F16.2)

      DATA EOF/3HEOF/

      IF (IVERT.NE.1) GO TO 7

      WRITE (IS1) (EOF,I=1,8)

      END FILE IS1

      REWIND IS1

5     CALL SESPL1 (IOX,6,2)
5     CALLSESPL1(IOX,6,2)

      WRITE (IOX,1)

6     READ (IS1) TIME,ETA,Z,THETA,PB,BOWACC,ACC,FANPWR

      IF (TIME.EQ.EOF) GO TO 7
```

```
      WRITE (IOx,2) TIME,ETA,Z,THETA,PB,BOWACC,ACC,FANPWR

      LINX=LINX+1
      LINX=LINX+1

      IF (LINX.EQ.50) GO TO 5

      GO TO 6

7     IF (ILATRL.NE.1) GO TO 10

      WRITE (IS2) (EOF,I=1,8)

      END FILE IS2

      REWIND IS2

8     CALL SESPL1 (IOX,6,2)
8     CALLSESPL1(IOX,6,2)

      WRITE (IOX,3)

9     READ (IS2) TIME,PHI,BETAS,ACCLAT,U,TRADUS,R,RUDANG

      IF (TIME.EQ.EOF) GO TO 10

      WRITE (IOX,4) TIME,PHI,BETAS,ACCLAT,U,TRADUS,R,RUDANG

      LINX=LINX+1
      LINX=LINX+1

      IF (LINX.EQ.50) GO TO 8

      GO TO 9

10    REWIND IS1

      REWIND IS2

      RETURN

      END
```

SYMBOL TABLE FOR MODULE COLFLA

VARIABLES

NAME	TYPE	RELOCATION
IVERT	INTEGR	COLUMN
ILATRL	INTEGR	COLUMN
IOX	INTEGR	SESCOM
LINX	INTEGR	SESCOM
IS1	INTEGR	UNITS
IS2	INTEGR	UNITS
EOF	REAL	
I	INTEGR	
TIME	REAL	
ETA	REAL	
Z	REAL	
THETA	REAL	
PB	REAL	
BOWACC	REAL	
ACC	REAL	
FANPWR	REAL	
PHI	REAL	
BETAS	REAL	
ACCLAT	REAL	
U	REAL	
TRADUS	REAL	
R	REAL	
RUDANG	REAL	

EXTERNALS

NAME	TYPE	ARGS
SESPL1		3

STATEMENT LABELS

1	2	3	4	7
5	6	10	8	9

COMMON BLOCKS

NAME	LENGTH
COLUMN	2
SESCOM	25
UNITS	4

****

COMMON BLOCK UNITS HAS INCORRECT SIZE

****

```
****** RESULTS OF FLOW ANALYSIS ******

      NO ERRORS FOUND


      NUMBER OF PATHS CHECKED-      4

                                          FLOW ANALYSIS TOOK      .050 CP SECONDS
```

Audit and Flow Analysis of Extraordinary Subroutine INOUT

```
      SUBROUTINE INOUT

      COMMON /SESCOM/ CASE(13),INA,INB,INC,IOX,NPAGX,LINX,IOY,NPAGY,LINY,IOZ,NPAGZ,LINZ


      COMMON /UNITS/ IS1,IS2,IS3,IS4

      INA=5
      INA=5

      INB=10
      INB=10

      IOX=6
      IOX=6

      IOY=6
      IOY=6

      IOZ=6
      IOZ=6

      IS1=1
      IS1=1

      IS2=2
      IS2=2

      IS3=44
      IS3=44

      IS4=3
      IS4=3

      RETURN

      END
```

SYMBOL TABLE FOR MODULE INOUT

VARIABLES

NAME	TYPE	RELOCATION
INA	INTEGR	SESCOM
INB	INTEGR	SESCOM
IOX	INTEGR	SESCOM
IOY	INTEGR	SESCOM
IOZ	INTEGR	SESCOM
IS1	INTEGR	UNITS
IS2	INTEGR	UNITS
IS3	INTEGR	UNITS
IS4	INTEGR	UNITS

COMMON BLOCKS

NAME	LENGTH
SESCOM	25
UNITS	4

****

COMMON BLOCK UNITS HAS INCORRECT SIZE

****

****** RESULTS OF FLOW ANALYSIS ******

NO ERRORS FOUND

NUMBER OF PATHS CHECKED- 1

FLOW ANALYSIS TOOK 3.000 CP SECONDS

Audit and Flow Analysis of Extraordinary Subroutine START

```
      SUBROUTINE START

      COMMON /USED01/ MUS01

      COMMON /USED02/ MUS02

      COMMON /USED03/ MUS03

      COMMON /USED04/ MUS04

      COMMON /USED06/ MUS06

      COMMON /USED08/ MUS08

      COMMON /USED10/ MUS10

      COMMON /USED11/ MUS11

      COMMON /USED14/ MUS14

      COMMON /USED17/ MUS17

      COMMON /USED18/ MUS18

      COMMON /USED19/ MUS19

      COMMON /USED20/ MUS20

      COMMON /PAGE1/ NUSE,NROLLC

      NUSE=0
      NUSE=0

      NROLLC=0
      NROLLC=0

      MUS01=0
      MUS01=0

      MUS02=0
      MUS02=0
```

```
MUS03=0
MUS03=0

MUS04=0
MUS04=0

MUS06=0
MUS06=0

MUS08=0
MUS08=0

MUS10=0
MUS10=0

MUS11=0
MUS11=0

MUS14=0
MUS14=0

MUS17=0
MUS17=0

MUS18=0
MUS18=0

MUS19=0
MUS19=0

MUS20=0
MUS20=0

RETURN

END
```

```
                    SYMBOL TABLE FOR MODULE START

                              VARIABLES

NAME            TYPE                            RELOCATION
MUS01            INTEGR                         USED01
MUS02            INTEGR                         USED02
MUS03            INTEGR                         USED03
MUS04            INTEGR                         USED04
MUS06            INTEGR                         USED06
MUS08            INTEGR                         USED08
MUS11            INTEGR                         USED11
MUS14            INTEGR                         USED14
MUS17            INTEGR                         USED17
MUS18            INTEGR                         USED18
MUS19            INTEGR                         USED19
MUS20            INTEGR                         USED20
NUSE             INTEGR                         PAGE1
NROLLC           INTEGR                         PAGE1

                          COMMON BLOCKS
                    NAME          LENGTH
                    USED01              1
                    USED02              1
                    USED03              1
                    USED04              1
                    USED06              1
                    USED08              1
                    USED10              1
                    USED11              1
                    USED14              1
                    USED17              1
                    USED18              1
                    USED19              1
                    USED20              1
                    PAGE1               2


**************************************************************************
     THE COMMON BLOCK SESCOM DOES NOT APPEAR IN THIS PROGRAM
**************************************************************************



****** RESULTS OF FLOW ANALYSIS ******


     NO ERRORS FOUND



     NUMBER OF PATHS CHECKED-     1

                                        FLOW ANALYSIS TOOK     .023 CP SECONDS
```

Audit of a BLOCK DATA Subprogram

```
BLOCK DATA
COMMON /AERODI/ Z1(21)
COMMON /AEROSI/ Z50(8)
COMMON /APPNDG/ Z61(262)
COMMON /APNDGO/ Z5E(EC)
COMMON /BMCO/ Z2(2)
COMMON /BOWSLI/ Z59(E9)
COMMON /BOWSLO/ Z54(40)
COMMON /CGLOC/ Z57(2)
COMMON /COLUMN/ Z3(2)
COMMON /EQNCO/ Z7(41)
COMMON /ENGINI/ Z45(1507)
COMMON /FANI/ Z46(354)
COMMON /FLAGS/ IJOB,ICASE,ISTEP,IERR
COMMON /FROUDE/ Z14(2)
COMMON /GBOW/ Z30(3)
COMMON /HELMS/ Z62(409)
COMMON /LEAKER/ Z22
COMMON /LOADS/ XMI
COMMON /MASSES/ Z23(812)
```

```
      COMMON /MATRIX/ Z24(36)
      COMMON /MSIDW/ Z25(25)
      COMMON /MWAVE/ Z26(6)
      COMMON /OPTION/ Z27(3)
      COMMON /PHYCON/ G,RHO,HRHO,ENU,RHOINF,PINF,GAM
      COMMON /PRINT/ ISES,IINTGR,IRHS
      COMMON /PROPI/ Z29(8)
      COMMON /SESCOM/ CASE(13),INA,INB,INC,IOX,NPAGX,LINX,IOY,NPAGY,LINY,IOZ,NPAGZ,LINZ
      COMMON /SIDWLI/ Z31(27)
      COMMON /SLOPE/ Z36(2)
      COMMON /SPRAY/ Z52(55)
      COMMON /STNSLI/ Z38(30)
      COMMON /STNSLO/ Z55(40)
      COMMON /TIMES/ Z33(5)
      COMMON /TORQUE/ Z34(27)
      COMMON /VARBLE/ Z41(15)
      COMMON /WAVE/ Z42(6)
      COMMON /WAVESI/ Z60(79)
      DATA Z1/21*0.0/
      DATA Z2/2*0.0/
      DATA Z3/2*0.0/
      DATA Z7/41*0.0/
```

```
      DATA Z14/2*0.0/
      DATA Z22/0.0/
      DATA Z23/812*0.0/
      DATA Z24/36*0.0/
      DATA Z25/29*0.0/
      DATA Z26/6*0.0/
      DATA Z27/3*0.0/
      DATA Z29/1.E6,7*0.0/
      DATA Z30/0.0/
**********************************************************************
     LIST SIZES DO NOT MATCH
**********************************************************************
      DATA Z31/27*0.0/
      DATA Z33/5*0.0/
      DATA Z34/27*0.0/
      DATA Z36/2*0.0/
      DATA Z38/30*0.0/
      DATA Z41/15*0.0/
      DATA Z42/6*0.0/
      DATA Z45/1507*0.0/
      DATA Z46/354*0.0/
      DATA Z50/8*0.0/
      DATA Z54/40*0.0/
      DATA Z55/40*0.0/
```

```
      DATA Z56/60*0.0/

      DATA Z57/2*0.0/

      DATA Z59/59*0.0/

      DATA Z60/79*0.0/

      DATA Z61/262*0.0/

      DATA Z62/409*0.0/

      DATA ENU,PINF,RHOINF,GAM/1.28E-5,2116.,.002378,1.4/

      DATA CASE/13*4H

      DATA INA,INB,INC,IOX,NPAGX,LINX,IOY,NPAGY,LINY,IOZ,NPAGZ,LINZ/12*0/

      DATA IJOB,ICASE,ISTEP,IERR/4*0/

      DATA XMI/0.0/

      DATA ISES,IINTGR,IRHS/0,0,0/

      END
```

SYMBOL TABLE FOR BLOCK DATA

VARIABLES

NAME	TYPE		RELOCATION
Z1	REAL	ARRAY 1	AERODI
Z50	REAL	ARRAY 1	AEROSI
Z61	REAL	ARRAY 1	APPNDG
Z56	REAL	ARRAY 1	APNDGO
Z2	REAL	ARRAY 1	BMCO
Z59	REAL	ARRAY 1	BOWSLI
Z54	REAL	ARRAY 1	BOWSLO
Z57	REAL	ARRAY 1	CGLOC
Z3	REAL	ARRAY 1	COLUMN
Z7	REAL	ARRAY 1	EQNCO
Z45	REAL	ARRAY 1	ENGINI
Z46	REAL	ARRAY 1	FANI
IJOB	INTEGR		FLAGS
ICASE	INTEGR		FLAGS
ISTEP	INTEGR		FLAGS
IERR	INTEGR		FLAGS
Z14	REAL	ARRAY 1	FROUDE
Z30	REAL	ARRAY 1	GBOW
Z62	REAL	ARRAY 1	HELMS
Z22	REAL		LEAKER
XMI	REAL		LOADS

Z23	REAL	ARRAY 1	MASSES
Z24	REAL	ARRAY 1	MATRIX
Z25	REAL	ARRAY 1	MSIDW
Z26	REAL	ARRAY 1	MWAVE
Z27	REAL	ARRAY 1	OPTION
ENU	REAL		PHYCON
RHOINF	REAL		PHYCON
PINF	REAL		PHYCON
GAM	REAL		PHYCON
ISES	INTEGR		PRINT
IINTGR	INTEGR		PRINT
IRHS	INTEGR		PRINT
Z29	REAL	ARRAY 1	PROPI
CASE	REAL	ARRAY 1	SESCOM
INA	INTEGR		SESCOM
INB	INTEGR		SESCOM
INC	INTEGR		SESCOM
IOX	INTEGR		SESCOM
NPAGX	INTEGR		SESCOM
LINX	INTEGR		SESCOM
IOY	INTEGR		SESCOM
NPAGY	INTEGR		SESCOM
LINY	INTEGR		SESCOM
IOZ	INTEGR		SESCOM
NPAGZ	INTEGR		SESCOM
LINZ	INTEGR		SESCOM
Z31	REAL	ARRAY 1	SIDWLI
Z36	REAL	ARRAY 1	SLOPE
Z38	REAL	ARRAY 1	STNSLI
Z55	REAL	ARRAY 1	STNSLO
Z33	REAL	ARRAY 1	TIMES
Z34	REAL	ARRAY 1	TORQUE
Z41	REAL	ARRAY 1	VARBLE
Z42	REAL	ARRAY 1	WAVE
Z60	REAL	ARRAY 1	WAVESI

COMMON BLOCKS

NAME	LENGTH
AERODI	21
AEROSI	8
APPNDG	262
APNDGO	60
BMCO	2
BOWSLI	59
BOWSLO	40
CGLOC	2
COLUMN	2
EQNCO	41
ENGINI	1507
FANI	354
FLAGS	4
FRCUDF	2
GROW	3
HELMS	409
LEAKER	1
LOADS	1
MASSES	812
MATRIX	36
MSIDW	25
MWAVE	6
OPTION	3
PHYCON	7
PRINT	3
PROPI	8
SESCOM	25
SIDWLI	27
SLOPE	2
SPRAY	55
STNSLI	30
STNSLO	40
TIMES	5
TORQUE	27
VARBLE	15
WAVE	6
WAVESI	79

```
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK AERODI DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK AEROSI DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK APPNDG DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK BMCO   DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK BOWSLI DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK COLUMN DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK FONCO  DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK ENGINI DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK FANI   DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK HELMS  DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      COMMON BLOCK MASSES HAS INCORRECT SIZE
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK OPTION DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK PROPI  DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK SIDWLI DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK SPRAY  DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK STNSLI DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
**********************************************************************************************
      WARNING - VARIABLE TYPE IN COMMON BLOCK WAVESI DOES NOT AGREE WITH INTERFACE DEFINITION
**********************************************************************************************
```

```
      PROGRAM MICRO(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)

   10 FORMAT(1H1,10H TRAPEZOID,6X,4HAREA,12X,5HERROR)

      INTEGER OUT

      OUT=6
GRAPH HAS BEEN PLACED INTO MEMORY

      OUT=6

      WRITE(OUT,10)

      DO 30 I=1,15

      N=2**(I-1)
      N=2**(I-1)

      A=0.0
      A=0.0

      W=1./FLOAT(N)
      W=Q1REAL(1./FLOAT(N))

      DO 20 J=1,N

      XL=FLOAT(J-1)*W
      XL=Q1REAL(FLOAT(J-1)*W)

   20 A=A+(W/2.)*(FUN(XL)+FUN(W+XL))
*******************************************************************************
     WARNING - THIS MODULE IS NOT IN THE SESCOMP LIST
*******************************************************************************
   20 A=Q1REAL(A+Q1REAL((Q1REAL(W/2.))*(Q1REAL(FUN(XL)+FUN(Q1REAL(W+XL))
  ))))

      E=5.-A
      E=Q1REAL(5.-A)

   30 WRITE(OUT,40)N,A,E

   40 FORMAT(I8,F16.8,E19.8)
*******************************************************************************
     THIS STATEMENT IS OUT OF ORDER
*******************************************************************************

      STOP

      END
*******************************************************************************
     WARNING - THIS MODULE IS NOT IN THE SESCOMP LIST
*******************************************************************************
```

SYMBOL TABLE FOR MODULE MICRO

VARIABLES

NAME	TYPE	RELOCATION
OUT	INTEGR	
I	INTEGR	
N	INTEGR	
A	REAL	
W	REAL	
J	INTEGR	
XL	REAL	
E	REAL	

EXTERNALS

NAME	TYPE	ARGS
FLOAT	REAL	1
FUN	REAL	1

STATEMENT LABELS

10 30 20 40

****

THE COMMON BLOCK SESCOM DOES NOT APPEAR IN THIS PROGRAM

****

****** RESULTS OF FLOW ANALYSIS ******

NO ERRORS FOUND

NUMBER OF PATHS CHECKED- 1

FLOW ANALYSIS TOOK 0.000 CP SECONDS

```
      FUNCTION FUN(X)

      FUN=6.-6.*X**5.
      FUN=Q1REAL(6.-Q1REAL(6.*Q1REAL(X**5.)))

      RETURN

      END
```

SYMBOL TABLE FOR MODULE FUN

VARIABLES

NAME	TYPE	RELOCATION
X	REAL	F. P.
FUN	REAL	

****
THE COMMON BLOCK SESCOM DOES NOT APPEAR IN THIS PROGRAM
****

****** RESULTS OF FLOW ANALYSIS ******

NO ERRORS FOUND

NUMBER OF PATHS CHECKED- 1

FLOW ANALYSIS TOOK 0.000 CP SECONDS

GLOBAL REFERENCE TABLE

EXTERNAL REFERENCES

FLOAT	ANSI FUNCTION
FUN	USER SUPPLIED

SUBROUTINES ENCOUNTERED

MICRO	USER SUPPLIED
FUN	USER SUPPLIED

```
      PROGRAM MICRO(INPUT,OUTPUT,TAPE5=INPUT,TAPE6=OUTPUT)
      COMPLEX Q1COMP
      DOUBLE PRECISION Q1DPRE
   10 FORMAT(1H1,10H TRAPEZOID,6X,4HAREA,12X,5HERROR)
      INTEGER OUT
      OUT=6
      WRITE(OUT,10)
      DO 30 I=1,15
      N=2**(I-1)
      A=0.0
      W=Q1REAL(1./FLOAT(N))
      DO 20 J=1,N
      XL=Q1REAL(FLOAT(J-1)*W)
   20 A=Q1REAL(A+Q1REAL((Q1REAL(W/2.))*(Q1REAL(FUN(XL)+FUN(Q1REAL(W+XL))
     *))))
      E=Q1REAL(5.-A)
   30 WRITE(OUT,40)N,A,E
   40 FORMAT(I8,F16.8,E19.8)
      STOP
      END
```

```
      FUNCTION FUN(X)
      COMPLEX Q1COMP
      DOUBLE PRECISION Q1DPRE
      FUN=Q1REAL(6.-Q1REAL(6.*Q1REAL(X**5.)))
      RETURN
      END
```

40-Bit Output

TRAPEZOID	AREA	ERROR
1	3.00000000	.20000000E+01
2	4.40625000	.59375000E+00
4	4.84570313	.15429688E+00
8	4.96105957	.38940430E-01
16	4.99024200	.97579956E-02
32	4.99755874	.24412572E-02
64	4.99938920	.61079860E-03
128	4.99984625	.15375018E-03
256	4.99995980	.40283333E-04
512	4.99998641	.13589859E-04
1024	4.99998930	.10699034E-04
2048	4.99998245	.17553568E-04
4096	4.99996680	.33199787E-04
8192	4.99993381	.66190958E-04
16384	4.99986607	.13393164E-03

39-Bit Output

TRAPEZOID	AREA	ERROR
1	3.00000000	.20000000E+01
2	4.40625000	.59375000E+00
4	4.84570313	.15429688E+00
8	4.96105957	.38940430E-01
16	4.99024200	.97579956E-02
32	4.99755847	.24415255E-02
64	4.99938852	.61148405E-03
128	4.99984539	.15461445E-03
256	4.99995804	.41961670E-04
512	4.99998242	.17583370E-04
1024	4.99998081	.19192696E-04
2048	4.99996537	.34630299E-04
4096	4.99993449	.65505505E-04
8192	4.99986660	.13339520E-03
16384	4.99973202	.26798248E-03

38-Bit Output

TRAPEZOID	AREA	ERROR
1	3.00000000	.20000000E+01
2	4.40625000	.59375000E+00
4	4.84570313	.15429688E+00
8	4.96105957	.38940430E-01
16	4.99024165	.97583532E-02
32	4.99755800	.24420023E-02
64	4.99938738	.61261654E-03
128	4.99984336	.15664101E-03
256	4.99995422	.45776367E-04
512	4.99997425	.25749207E-04
1024	4.99996316	.36835670E-04
2048	4.99993193	.68068504E-04
4096	4.99986589	.13411045E-03
8192	4.99973035	.26965141E-03
16384	4.99946296	.53703785E-03

37-Bit Output

TRAPEZOID	AREA	ERROR
1	3.00000000	.20000000E+01
2	4.40625000	.59375000E+00
4	4.84570313	.15429688E+00
8	4.96105957	.38940430E-01
16	4.99024105	.97589493E-02
32	4.99755692	.24430752E-02
64	4.99938440	.61559677E-03
128	4.99983954	.16045570E-03
256	4.99994540	.54597855E-04
512	4.99995756	.42438507E-04
1024	4.99993038	.69618225E-04
2048	4.99986792	.13208389E-03
4096	4.99973464	.26535988E-03
8192	4.99945807	.54192543E-03
16384	4.99891925	.10807514E-02

36-Bit Output

TRAPEZOID	AREA	ERROR
1	3.00000000	.20000000E+01
2	4.40625000	.59375000E+00
4	4.84570313	.15429688E+00
8	4.96105957	.38940430E-01
16	4.99024010	.97599030E-02
32	4.99755526	.24447441E-02
64	4.99938154	.61845779E-03
128	4.99983263	.16736984E-03
256	4.99992990	.70095062E-04
512	4.99992371	.76293945E-04
1024	4.99986124	.13875961E-03
2048	4.99973631	.26369095E-03
4096	4.99947166	.52833557E-03
8192	4.99891567	.10843277E-02
16384	4.99783564	.21643639E-02

35-Bit Output

TRAPEZOID	AREA	ERROR
1	3.00000000	.20000000E+01
2	4.40625000	.59375000E+00
4	4.84570313	.15429688E+00
8	4.96105957	.38940430E-01
16	4.99023724	.97627640E-02
32	4.99755096	.24490356E-02
64	4.99937153	.62847137E-03
128	4.99981785	.18215179E-03
256	4.99989986	.10013580E-03
512	4.99985313	.14686584E-03
1024	4.99971485	.28514862E-03
2048	4.99947166	.52833557E-03
4096	4.99892998	.10700226E-02
8192	4.99783134	.21686554E-02
16384	4.99563503	.43649673E-02

34-Bit Output

TRAPEZOID	AREA	ERROR
1	3.00000000	.20000000E+01
2	4.40625000	.59375000E+00
4	4.84570313	.15429688E+00
8	4.96105957	.38940430E-01
16	4.99023247	.97675323E-02
32	4.99753952	.24604797E-02
64	4.99935532	.64468384E-03
128	4.99978828	.21171570E-03
256	4.99983406	.16593933E-03
512	4.99972725	.27275085E-03
1024	4.99944305	.55694580E-03
2048	4.99892235	.10776520E-02
4096	4.99784660	.21533966E-02
8192	4.99562836	.43716431E-02
16384	4.99120522	.87947845E-02

33-Bit Output

TRAPEZOID	AREA	ERROR
1	3.00000000	.20000000E+01
2	4.40625000	.59375000E+00
4	4.84570313	.15429688E+00
8	4.96105957	.38940430E-01
16	4.99022675	.97732544E-02
32	4.99752426	.24757385E-02
64	4.99932480	.67520142E-03
128	4.99972534	.27465820E-03
256	4.99969482	.30517578E-03
512	4.99943542	.56457520E-03
1024	4.99889755	.11024475E-02
2048	4.99784470	.21553040E-02
4096	4.99562073	.43792725E-02
8192	4.99120712	.87928772E-02
16384	4.98219299	.17807007E-01

32-Bit Output

TRAPEZOID	AREA	ERROR
1	3.00000000	.20000000E+01
2	4.40625000	.59375000E+00
4	4.84570313	.15429688E+00
8	4.96105194	.38948059E-01
16	4.99021149	.97885132E-02
32	4.99747467	.25253296E-02
64	4.99925232	.74768066E-03
128	4.99957275	.42724609E-03
256	4.99941254	.58746338E-03
512	4.99894714	.10528564E-02
1024	4.99781799	.21820068E-02
2048	4.99563599	.43640137E-02
4096	4.99114990	.88500977E-02
8192	4.98219299	.17807007E-01
16384	4.96421051	.35789490E-01

31-Bit Output

TRAPEZOID	AREA	ERROR
1	3.00000000	.20000000E+01
2	4.40625000	.59375000E+00
4	4.84570313	.15429688E+00
8	4.96101379	.38986206E-01
16	4.99017334	.98266602E-02
32	4.99740601	.25939941E-02
64	4.99908447	.91552734E-03
128	4.99931335	.68664551E-03
256	4.99894714	.10528564E-02
512	4.99781799	.21820068E-02
1024	4.99565125	.43487549E-02
2048	4.99116516	.88348389E-02
4096	4.98208618	.17913818E-01
8192	4.96430969	.35690308E-01
16384	4.92608643	.73913574E-01

30-Bit Output

TRAPEZOID	AREA	ERROR
1	3.00000000	.20000000E+01
2	4.40625000	.59375000E+00
4	4.84570313	.15429688E+00
8	4.96099854	.39001465E-01
16	4.99014282	.98571777E-02
32	4.99725342	.27465820E-02
64	4.99884033	.11596680E-02
128	4.99880981	.11901855E-02
256	4.99789429	.21057129E-02
512	4.99542236	.45776367E-02
1024	4.99136353	.86364746E-02
2048	4.98220825	.17791748E-01
4096	4.96411133	.35888672E-01
8192	4.92617798	.73822021E-01
16384	4.84579468	.15420532E+00

INITIAL DISTRIBUTION

Copies

10	NAVSEA PMS304-32 White
2	NAVSEA PMS405-40 Cuthbert
12	DDC

CENTER DISTRIBUTION

1	18/1809
1	1802.2 Frenkiel
1	1802.4 Theilheimer
1	1809.3 D. Harris (Central Depository, CMLD)
1	182 Camara
1	1826 Culpepper
30	1826 Wybraniec
1	184 Lugt
1	185 Corin
1	186 Sulit
1	189 Gray
1	1890 Taylor
30	5214.1 Reports Distribution
1	522

Microfiche copies

30	1826 Wybraniec